Using Driverless AI

Release 1.6.2

H2O.ai

May 13, 2019
7.2 Setting Variables in Native Installs ........................................... 141

8 Enabling Data Connectors ......................................................... 143
  8.1 Using Data Connectors with the Docker Image .......................... 143
  8.2 Using Data Connectors with Native Installs ............................ 159

9 Configuring Authentication ..................................................... 181
  9.1 Local Authentication Example ............................................. 181
  9.2 LDAP Authentication Example ............................................ 183
  9.3 PAM Authentication Example ............................................. 186

10 Enabling Notifications ......................................................... 189
  10.1 Script Interfaces ............................................................. 189
  10.2 Example ................................................................. 189

11 Launching Driverless AI .......................................................... 193
  11.1 Resources ................................................................. 194
  11.2 Messages ................................................................. 194

12 The Datasets Page .................................................................. 195
  12.1 Supported File Types ....................................................... 195
  12.2 Adding Datasets .............................................................. 196
  12.3 Dataset Details ............................................................... 198
  12.4 Splitting Datasets ............................................................ 200
  12.5 Visualizing Datasets ......................................................... 202

13 Experiments ......................................................................... 207
  13.1 Before You Begin .............................................................. 207
  13.2 New Experiments ............................................................. 233
  13.3 Experiment Graphs ........................................................... 237
  13.4 Completed Experiment ...................................................... 241
  13.5 Viewing Experiments ......................................................... 245

14 Diagnosing a Model ............................................................... 249

15 Interpreting a Model ............................................................. 253
  15.1 Additional Resources ......................................................... 253
  15.2 Intrepret this Model Button - Non-Time-Series ...................... 253
  15.3 Interpret this Model Button - Time-Series ............................. 254
  15.4 Model Interpretation on Driverless AI Models ....................... 257
  15.5 Model Interpretation on External Models ............................. 259
  15.6 Understanding the Model Interpretation Page ......................... 261
  15.7 General Considerations ..................................................... 271
  15.8 Current Known Issues ........................................................ 272

16 Viewing Explanations ............................................................. 273

17 Score on Another Dataset ...................................................... 277

18 Transform Another Dataset .................................................... 279

19 The Python and MOJO Scoring Pipelines ................................. 281

20 Which Pipeline Should I Use? .................................................. 283
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>21</td>
<td>Driverless AI Standalone Python Scoring Pipeline</td>
<td>285</td>
</tr>
<tr>
<td>21.1</td>
<td>Python Scoring Pipeline Files</td>
<td>285</td>
</tr>
<tr>
<td>21.2</td>
<td>Quick Start</td>
<td>285</td>
</tr>
<tr>
<td>21.3</td>
<td>The Python Scoring Module</td>
<td>289</td>
</tr>
<tr>
<td>21.4</td>
<td>The Scoring Service</td>
<td>290</td>
</tr>
<tr>
<td>21.5</td>
<td>Python Scoring Pipeline FAQ</td>
<td>292</td>
</tr>
<tr>
<td>21.6</td>
<td>Troubleshooting Python Environment Issues</td>
<td>293</td>
</tr>
<tr>
<td>22</td>
<td>Driverless AI MLI Standalone Python Scoring Package</td>
<td>295</td>
</tr>
<tr>
<td>22.1</td>
<td>MLI Python Scoring Package Files</td>
<td>295</td>
</tr>
<tr>
<td>22.2</td>
<td>Quick Start</td>
<td>295</td>
</tr>
<tr>
<td>22.3</td>
<td>MLI Python Scoring Module</td>
<td>298</td>
</tr>
<tr>
<td>22.4</td>
<td>K-LIME vs Shapley Reason Codes</td>
<td>299</td>
</tr>
<tr>
<td>22.5</td>
<td>MLI Scoring Service Overview</td>
<td>299</td>
</tr>
<tr>
<td>23</td>
<td>Driverless AI MOJO Scoring Pipeline</td>
<td>303</td>
</tr>
<tr>
<td>23.1</td>
<td>Prerequisites</td>
<td>303</td>
</tr>
<tr>
<td>23.2</td>
<td>Enabling the MOJO Scoring Pipeline</td>
<td>304</td>
</tr>
<tr>
<td>23.3</td>
<td>MOJO Scoring Pipeline Files</td>
<td>304</td>
</tr>
<tr>
<td>23.4</td>
<td>Quickstart</td>
<td>304</td>
</tr>
<tr>
<td>23.5</td>
<td>Compile and Run the MOJO from Java</td>
<td>305</td>
</tr>
<tr>
<td>23.6</td>
<td>Using the MOJO Scoring Pipeline with Spark/Sparkling Water</td>
<td>307</td>
</tr>
<tr>
<td>24</td>
<td>Deploying the MOJO Pipeline</td>
<td>311</td>
</tr>
<tr>
<td>24.1</td>
<td>Deployments Overview Page</td>
<td>311</td>
</tr>
<tr>
<td>24.2</td>
<td>Amazon Lambda Deployment</td>
<td>311</td>
</tr>
<tr>
<td>24.3</td>
<td>AWS Deployment Issues</td>
<td>317</td>
</tr>
<tr>
<td>25</td>
<td>What’s Happening in Driverless AI?</td>
<td>319</td>
</tr>
<tr>
<td>26</td>
<td>Data Sampling</td>
<td>321</td>
</tr>
<tr>
<td>27</td>
<td>Driverless AI Transformations</td>
<td>325</td>
</tr>
<tr>
<td>27.1</td>
<td>Available Transformers</td>
<td>325</td>
</tr>
<tr>
<td>27.2</td>
<td>Example Transformations</td>
<td>329</td>
</tr>
<tr>
<td>28</td>
<td>Internal Validation Technique</td>
<td>333</td>
</tr>
<tr>
<td>29</td>
<td>Time Series in Driverless AI</td>
<td>335</td>
</tr>
<tr>
<td>29.1</td>
<td>Understanding Time Series</td>
<td>335</td>
</tr>
<tr>
<td>29.2</td>
<td>Time Series Constraints</td>
<td>340</td>
</tr>
<tr>
<td>29.3</td>
<td>Time Series Use Case: Sales Forecasting</td>
<td>340</td>
</tr>
<tr>
<td>29.4</td>
<td>Time Series EXPERT SETTINGS</td>
<td>342</td>
</tr>
<tr>
<td>29.5</td>
<td>Using a Driverless AI Time Series Model to Forecast</td>
<td>343</td>
</tr>
<tr>
<td>30</td>
<td>NLP in Driverless AI</td>
<td>345</td>
</tr>
<tr>
<td>30.1</td>
<td>A Typical NLP Example: Sentiment Analysis</td>
<td>345</td>
</tr>
<tr>
<td>31</td>
<td>Driverless AI Logs</td>
<td>349</td>
</tr>
<tr>
<td>31.1</td>
<td>Accessing Driverless AI Logs</td>
<td>349</td>
</tr>
<tr>
<td>31.2</td>
<td>Sending Logs to H2O</td>
<td>354</td>
</tr>
<tr>
<td>32</td>
<td>FAQ</td>
<td>355</td>
</tr>
<tr>
<td>32.1</td>
<td>General</td>
<td>357</td>
</tr>
<tr>
<td>32.2</td>
<td>Installation/Upgrade/Authentication</td>
<td>358</td>
</tr>
<tr>
<td>32.3</td>
<td>Data/Experiments/Predictions</td>
<td>362</td>
</tr>
</tbody>
</table>
H2O Driverless AI is an artificial intelligence (AI) platform for automatic machine learning. Driverless AI automates some of the most difficult data science and machine learning workflows such as feature engineering, model validation, model tuning, model selection and model deployment. It aims to achieve highest predictive accuracy, comparable to expert data scientists, but in much shorter time thanks to end-to-end automation. Driverless AI also offers automatic visualizations and machine learning interpretability (MLI). Especially in regulated industries, model transparency and explanation are just as important as predictive performance. Modeling pipelines (feature engineering and models) are exported (in full fidelity, without approximations) both as Python modules and as pure Java standalone scoring artifacts.

Driverless AI runs on commodity hardware. It was also specifically designed to take advantage of graphical processing units (GPUs), including multi-GPU workstations and servers such as IBM’s Power9-GPU AC922 server and the NVIDIA DGX-1 for order-of-magnitude faster training.

This document describes how to install and use Driverless AI. For more information about Driverless AI, please see https://www.h2o.ai/products/h2o-driverless-ai/.

For a third-party review, please see https://www.infoworld.com/article/3236048/machine-learning/review-h2oai-automates-machine-learning.html.

Have Questions?

If you have questions about using Driverless AI, post them on Stack Overflow using the driverless-ai tag at http://stackoverflow.com/questions/tagged/driverless-ai.
H2O DRIVERLESS AI RELEASE NOTES

H2O Driverless AI is a high-performance, GPU-enabled, client-server application for the rapid development and deployment of state-of-the-art predictive analytics models. It reads tabular data from various sources and automates data visualization, grand-master level automatic feature engineering, model validation (overfitting and leakage prevention), model parameter tuning, model interpretability and model deployment. H2O Driverless AI is currently targeting common regression, binomial classification, and multinomial classification applications including loss-given-default, probability of default, customer churn, campaign response, fraud detection, anti-money-laundering, and predictive asset maintenance models. It also handles time-series problems for individual or grouped time-series such as weekly sales predictions per store and department, with time-causal feature engineering and validation schemes. The ability to model unstructured data is coming soon.

High-level capabilities:

• Client/server application for rapid experimentation and deployment of state-of-the-art supervised machine learning models
• Automatically creates machine learning modeling pipelines for highest predictive accuracy
• Automatically creates stand-alone scoring pipeline for in-process scoring or client/server scoring via http or tcp protocols, in Python and Java (low-latency scoring).
• Python API or GUI (Java API coming soon)
• Multi-GPU and multi-CPU support for powerful workstations and NVidia DGX supercomputers
• Machine Learning model interpretation module with global and local model interpretation
• Automatic Visualization module
• Multi-user support
• Backward compatibility

Problem types supported:

• Regression (continuous target variable, for age, income, house price, loss prediction, time-series forecasting)
• Binary classification (0/1 or “N”/”Y”, for fraud prediction, churn prediction, failure prediction, etc.)
• Multinomial classification (0/1/2/3 or “A”/”B”/”C”/”D” for categorical target variables, for prediction of membership type, next-action, product recommendation, etc.)

Data types supported:

• Tabular structured data, rows are observations, columns are fields/features/variables
• i.i.d. (identically and independently distributed) data
• Numeric, categorical and textual fields
• Missing values are allowed
• Time-series data with a single time-series (time flows across the entire dataset, not per block of data)
• Grouped time-series (e.g., sales per store per department per week, all in one file, with 3 columns for store, dept, week)
• Time-series problems with a gap between training and testing (i.e., the time to deploy), and a known forecast horizon (after which model has to be retrained)

Data types NOT supported:
• Image/video/audio

Data sources supported:
• Local file system or NFS
• File upload from browser or Python client
• Hadoop (HDFS)
• S3 (Amazon)
• Azure Blob storage
• Blue Data Tap
• Google big query
• Google cloud storage
• kdb+
• Minio
• Snowflake

File formats supported:
• Plain text formats of columnar data (.csv, .tsv, .txt)
• Compressed archives (.zip, .gz, .bz2)
• Excel
• Parquet
• Feather
• Python datatable (.nff, .jay)
1.1 Architecture

![DAI architecture diagram](image-url)

Fig. 1: DAI architecture
1.2 Roadmap

![Driverless AI Roadmap (Public)](image)

Fig. 2: DAI roadmap

1.3 Change Log

1.3.1 Version 1.6.2 LTS (May 10, 2019)

- This version provides PPC64le artifacts
- Improved stability of datatable
- Improved path filtering in the file browser
- Fixed units for RMSPE scorer to be a percentage (multiply by 100)
- Fixed segmentation fault on Ubuntu 18 with installed font package
- Fixed IBM Spectrum Conductor authentication
- Fixed handling of EC2 machine credentials
- Fixed of Lag transformer configuration
- Fixed KDB and Snowflake Error Reporting
- Gradually reduce number of used workers for column statistics computation in case of failure.
- Hide default Tornado header exposing used version of Tornado
- Documentation updates:
Using Driverless AI, Release 1.6.2

- Added instructions for installing via AWS Marketplace
- Improved documentation for installing via Google Cloud
- Improved FAQ documentation
- Added Data Sampling documentation topic
- Various bug fixes

### 1.3.2 Version 1.6.1.1 LTS (Apr 24, 2019)

Available here
- Fix in AWS role handling.

### 1.3.3 Version 1.6.1 LTS (Apr 18, 2019)

Available here
- Several fixes for MLI (partial dependence plots, Shapley values)
- Improved documentation for model deployment, time-series scoring, AutoVis and FAQs

### 1.3.4 Version 1.6.0 LTS (Apr 5, 2019)

Private build only.
- Fixed import of string columns larger than 2GB
- Fixed AutoViz crashes on Windows
- Fixed quantile binning in MLI
- Plot global absolute mean Shapley values instead of global mean Shapley values in MLI
- Improvements to PDP/ICE plots in MLI
- Validated Terraform version in AWS Lambda deployment
- Added support for NULL variable importance in AutoDoc
- Made Variable Importance table size configurable in AutoDoc
- Improved support for various combinations of data import options being enabled/disabled
- CUDA is now part of distribution for easier installation
- Security updates:
  - Enforced SSL settings to be honored for all h2oai_client calls
  - Added config option to prevent using LocalStorage in the browser to cache information
  - Upgraded Tornado server version to 5.1.1
  - Improved session expiration and autologout functionality
  - Disabled access to Driverless AI data folder in file browser
  - Provided an option to filter content that is shown in the file browser
  - Use login name for HDFS impersonation instead of predefined name
– Disabled autocomplete in login form
  • Various bug fixes

1.3.5 Version 1.5.4 (Feb 24, 2019)

Available here
  • Speed up calculation of column statistics for date/datetime columns using certain formats (now uses ‘max_rows_col_stats’ parameter)
  • Added computation of standard deviation for variable importances in experiment summary files
  • Added computation of shift of variable importances between feature evolution and final pipeline
  • Fix link to MLI Time-Series experiment
  • Fix display bug for iteration scores for long experiments
  • Fix display bug for early finish of experiment for GLM models
  • Fix display bug for k-LIME when target is skewed
  • Fix display bug for forecast horizon in MLI for Time-Series
  • Fix MLI for Time-Series for single time group column
  • Fix in-server scoring of time-series experiments created in 1.5.0 and 1.5.1
  • Fix OpenBLAS dependency
  • Detect disabled GPU persistence mode in Docker
  • Reduce disk usage during TensorFlow NLP experiments
  • Reduce disk usage of aborted experiments
  • Refresh reported size of experiments during start of application
  • Disable TensorFlow NLP transformers by default to speed up experiments (can enable in expert settings)
  • Improved progress percentage shown during experiment
  • Improved documentation (upgrade on Windows, how to create the simplest model, DTap connectors, etc.)
  • Various bug fixes

1.3.6 Version 1.5.3 (Feb 8, 2019)

Available here
  • Added support for splitting datasets by time via time column containing date, datetime or integer values
  • Added option to disable file upload
  • Require authentication to download experiment artifacts
  • Automatically drop predictor columns from training frame if not found in validation or test frame and warn
  • Improved performance by using physical CPU cores only (configurable in config.toml)
  • Added option to not show inactive data connectors
  • Various bug fixes
1.3.7 Version 1.5.2 (Feb 2, 2019)

Available here

• Added world-level bidirectional GRU Tensorflow models for NLP features
• Added character-level CNN Tensorflow models for NLP features
• Added support to import multiple individual datasets at once
• Added support for holdout predictions for time-series experiments
• Added support for regression and multinomial classification for FTRL (in addition to binomial classification)
• Improved scoring for time-series when test data contains actual target values (missing target values will be predicted)
• Reduced memory usage for LightGBM models
• Improved performance for feature engineering
• Improved speed for TensorFlow models
• Improved MLI GUI for time-series problems
• Fix final model fold splits when fold_column is provided
• Various bug fixes

1.3.8 Version 1.5.1 (Jan 22, 2019)

Available here

• Fix MOJO for GLM
• Add back .csv file of experiment summary
• Improve collection of pipeline timing artifacts
• Clean up Docker tag

1.3.9 Version 1.5.0 (Jan 18, 2019)

Available here

• Added model diagnostics (interactive model metrics on new test data incl. residual analysis for regression)
• Added FTRL model (Follow The Regularized Leader)
• Added Kolmogorov-Smirnov metric (degree of separation between positives and negatives)
• Added ability to retrain (only) the final model on new data
• Added one-hot encoding for low-cardinality categorical features, for GLM
• Added choice between 32-bit (now default) and 64-bit precision
• Added system information (CPU, GPU, disk, memory, experiments)
• Added support for time-series data with many more time gaps, and with weekday-only data
• Added one-click deployment to Amazon Lambda
• Added ability to split datasets randomly, with option to stratify by target column or group by fold column
• Added support for OpenID authentication
Using Driverless AI, Release 1.6.2

- Added connector for BlueData
- Improved responsiveness of the GUI under heavy load situations
- Improved speed and reduce memory footprint of feature engineering
- Improved performance for RuleFit models and enable GPU and multinomial support
- Improved auto-detection of temporal frequency for time-series problems
- Improved accuracy of final single model if external validation provided
- Improved final pipeline if external validation data is provided (add ensembling)
- Improved k-LIME in MLI by using original features deemed important by DAI instead of all original features
- Improved MLI by using 3-fold CV by default for all surrogate models
- Improved GUI for MLI time series (integrated help, better integration)
- Added ability to view MLI time series logs while MLI time series experiment is running
- PDF version of the Automatic Report (AutoDoc) is now replaced by a Word version
- Various bug fixes (GLM accuracy, UI slowness, MLI UI, AutoVis)

1.3.10 Version 1.4.2 (Dec 3, 2018)

Available here
- Support for IBM Power architecture
- Speed up training and reduce size of final pipeline
- Reduced resource utilization during training of final pipeline
- Display test set metrics (ROC, ROCPR, Gains, Lift) in GUI in addition to validation metrics (if test set provided)
- Show location of best threshold for Accuracy, MCC and F1 in ROC curves
- Add relative point sizing for scatter plots in AutoVis
- Fix file upload and add model checkpointing in python client API
- Various bug fixes

1.3.11 Version 1.4.1 (Nov 11, 2018)

Available here
- Improved integration of MLI for time-series
- Reduced disk and memory usage during final ensemble
- Allow scoring and transformations on previously imported datasets
- Enable checkpoint restart for unfinished models
- Add startup checks for OpenCL platforms for LightGBM on GPUs
- Improved feature importances for ensembles
- Faster dataset statistics for date/datetime columns
- Faster MOJO batch scoring
- Fix potential hangs
• Fix ‘not in list’ error in MOJO
• Fix NullPointerException in MLI
• Fix outlier detection in AutoVis
• Various bug fixes

1.3.12 Version 1.4.0 (Oct 27, 2018)

Available here

• Enable LightGBM by default (now with MOJO)
• LightGBM tuned for GBM decision trees, Random Forest (rf), and Dropouts meet Multiple Additive Regression Trees (dart)
• Add ‘isHoliday’ feature for time columns
• Add ‘time’ column type for date/datetime columns in data preview
• Add support for binary datatable file ingest in .jay format
• Improved final ensemble (each model has its own feature pipeline)
• Automatic smart checkpointing (feature brain) from prior experiments
• Add kdb+ connector
• Feature selection of original columns for data with many columns to handle >>100 columns
• Improved time-series recipe (multiple validation splits, better logic)
• Improved performance of AutoVis
• Improved date detection logic (now detects %Y%m%d and %Y-%m date formats)
• Automatic fallback to CPU mode if GPU runs out of memory (for XGBoost, GLM and LightGBM)
• No longer require header for validation and testing datasets if data types match
• No longer include text columns for data shift detection
• Add support for time-series models in MLI (including ability to select time-series groups)
• Add ability to download MLI logs from MLI experiment page (includes both Python and Java logs)
• Add ability to view MLI logs while MLI experiment is running (Python and Java logs)
• Add ability to download LIME and Shapley reason codes from MLI page
• Add ability to run MLI on transformed features
• Display all variables for MLI variable importance for both DAI and surrogate models in MLI summary
• Include variable definitions for DAI variable importance list in MLI summary
• Fix Gains/Lift charts when observations weights are given
• Various bug fixes
1.3.13 Version 1.3.1 (Sep 12, 2018)

Available here

- Fix ‘Broken pipe’ failures for TensorFlow models
- Fix time-series problems with categorical features and interpretability >= 8
- Various bug fixes

1.3.14 Version 1.3.0 (Sep 4, 2018)

Available here

- Added LightGBM models - now have [XGBoost, LightGBM, GLM, TensorFlow, RuleFit]
- Added TensorFlow NLP recipe based on CNN Deeplearning models (sentiment analysis, document classification, etc.)
- Added MOJO for GLM
- Added detailed confusion matrix statistics
- Added more expert settings
- Improved data exploration (columnar statistics and row-based data preview)
- Improved speed of feature evolution stage
- Improved speed of GLM
- Report single-pass score on external validation and test data (instead of bootstrap mean)
- Reduced memory overhead for data processing
- Reduced number of open files - fixes ‘Bad file descriptor’ error on Mac/Docker
- Simplified Python client API
- Query any data point in the MLI UI from the original dataset due to “on-demand” reason code generation
- Enhanced k-means clustering in k-LIME by only using a subset of features. See The K-LIME Technique for more information.
- Report k-means centers for k-LIME in MLI summary for better cluster interpretation
- Improved MLI experiment listing details
- Various bug fixes

1.3.15 Version 1.2.2 (July 5, 2018)

Available here

- MOJO Java scoring pipeline for time-series problems
- Multi-class confusion matrices
- AUCMACRO Scorer: Multi-class AUC via macro-averaging (in addition to the default micro-averaging)
- Expert settings (configuration override) for each experiment from GUI and client APIs.
- Support for HTTPS
- Improved downsampling logic for time-series problems (if enabled through accuracy knob settings)
• LDAP readonly access to Active Directory
• Snowflake data connector
• Various bug fixes

1.3.16 Version 1.2.1 (June 26, 2018)

• Added LIME-SUP (alpha) to MLI as alternative to k-LIME (local regions are defined by decision tree instead of k-means)
• Added RuleFit model (alpha), now have [GBM, GLM, TensorFlow, RuleFit] - TensorFlow and RuleFit are disabled by default
• Added Minio (private cloud storage) connector
• Added support for importing folders from S3
• Added ‘Upload File’ option to ‘Add Dataset’ (in addition to drag & drop)
• Predictions for binary classification problems now have 2 columns (probabilities per class), for consistency with multi-class
• Improved model parameter tuning
• Improved feature engineering for time-series problems
• Improved speed of MOJO generation and loading
• Improved speed of time-series related automatic calculations in the GUI
• Fixed potential rare hangs at end of experiment
• No longer require internet to run MLI
• Various bug fixes

1.3.17 Version 1.2.0 (June 11, 2018)

• Time-Series recipe
• Low-latency standalone MOJO Java scoring pipelines (now beta)
• Enable Elastic Net Generalized Linear Modeling (GLM) with lambda search (and GPU support), for interpretability>=6 and accuracy<=5 by default (alpha)
• Enable TensorFlow (TF) Deep Learning models (with GPU support) for interpretability=1 and/or multi-class models (alpha, enable via config.toml)
• Support for pre-tuning of [GBM, GLM, TF] models for picking best feature evolution model parameters
• Support for final ensemble consisting of mix of [GBM, GLM, TF] models
• Automatic Report (AutoDoc) in PDF and Markdown format as part of summary zip file
• Interactive tour (assistant) for first-time users
• MLI now runs on experiments from previous releases
• Surrogate models in MLI now use 3 folds by default
• Improved small data recipe with up to 10 cross-validation folds
• Improved accuracy for binary classification with imbalanced data
• Additional time-series transformers for interactions and aggregations between lags and lagging of non-target columns
• Faster creation of MOJOs
• Progress report during data ingest
• Normalize binarized multi-class confusion matrices by class count (global scaling factor)
• Improved parsing of boolean environment variables for configuration
• Various bug fixes

### 1.3.18 Version 1.1.6 (May 29, 2018)

• Improved performance for large datasets
• Improved speed and user interface for MLI
• Improved accuracy for binary classification with imbalanced data
• Improved generalization estimate for experiments with given validation data
• Reduced size of experiment directories
• Support for Parquet files
• Support for bzip2 compressed files
• Added Data preview in UI: ‘Describe’
• No longer add ID column to holdout and test set predictions for simplicity
• Various bug fixes

### 1.3.19 Version 1.1.4 (May 17, 2018)

• Native builds (RPM/DEB) for 1.1.3

### 1.3.20 Version 1.1.3 (May 16, 2018)

• Faster speed for systems with large CPU core counts
• Faster and more robust handling of user-specified missing values for training and scoring
• Same validation scheme for feature engineering and final ensemble for high enough accuracy
• MOJO scoring pipeline for text transformers
• Fixed single-row scoring in Python scoring pipeline (broken in 1.1.2)
• Fixed default scorer when experiment is started too quickly
• Improved responsiveness for time-series GUI
• Improved responsiveness after experiment abort
• Improved load balancing of memory usage for multi-GPU XGBoost
• Improved UI for selection of columns to drop
• Various bug fixes
1.3.21 Version 1.1.2 (May 8, 2018)

- Support for automatic time-series recipe (alpha)
- Now using Generalized Linear Model (GLM) instead of XGBoost (GBM) for interpretability
- Added experiment preview with runtime and memory usage estimation
- Added MER scorer (Median Error Rate, Median Abs. Percentage Error)
- Added ability to use integer column as time column
- Speed up type enforcement during scoring
- Support for reading ARFF file format (alpha)
- Quantile Binning for MLI
- Various bug fixes

1.3.22 Version 1.1.1 (April 23, 2018)

- Support string columns larger than 2GB

1.3.23 Version 1.1.0 (April 19, 2018)

- AWS/Azure integration (hourly cloud usage)
- Bug fixes for MOJO pipeline scoring (now beta)
- Google Cloud storage and BigQuery (alpha)
- Speed up categorical column stats computation during data import
- Further improved memory management on GPUs
- Improved accuracy for MAE scorer
- Ability to build scoring pipelines on demand (if not enabled by default)
- Additional target transformer for regression problems $\sqrt{\sqrt{x}}$
- Add GLM models as candidates for interpretability=10 (alpha, disabled by default)
- Improved performance of native builds (RPM/DEB)
- Improved estimation of error bars
- Various bug fixes

1.3.24 Version 1.0.30 (April 5, 2018)

- Speed up MOJO pipeline creation and disable MOJO by default (still alpha)
- Improved memory management on GPUs
- Support for optional 32-bit floating-point precision for reduced memory footprint
- Added logging of test set scoring and data transformations
- Various bug fixes
1.3.25 Version 1.0.29 (April 4, 2018)
• If MOJO fails to build, no MOJO will be available, but experiment can still succeed

1.3.26 Version 1.0.28 (April 3, 2018)
• (Non-docker) RPM installers for RHEL7/CentOS7/SLES 12 with systemd support

1.3.27 Version 1.0.27 (March 31, 2018)
• MOJO scoring pipeline for Java standalone cross-platform low-latency scoring (alpha)
• Various bug fixes

1.3.28 Version 1.0.26 (March 28, 2018)
• Improved performance and reduced memory usage for large datasets
• Improved performance for F0.5, F2 and accuracy
• Improved performance of MLI
• Distribution shift detection now also between validation and test data
• Batch scoring example using datatable
• Various enhancements for AutoVis (outliers, parallel coordinates, log file)
• Various bug fixes

1.3.29 Version 1.0.25 (March 22, 2018)
• New scorers for binary/multinomial classification: F0.5, F2 and accuracy
• Precision-recall curve for binary/multinomial classification models
• Plot of actual vs predicted values for regression problems
• Support for excluding feature transformations by operation type
• Support for reading binary file formats: datatable and Feather
• Improved multi-GPU memory load balancing
• Improved display of initial tuning results
• Reduced memory usage during creation of final model
• Fixed several bugs in creation of final scoring pipeline
• Various UI improvements (e.g., zooming on iteration scoreboard)
• Various bug fixes
1.3.30 Version 1.0.24 (March 8, 2018)

- Fix test set scoring bug for data with an ID column (introduced in 1.0.23)
- Allow renaming of MLI experiments
- Ability to limit maximum number of cores used for datatable
- Print validation scores and error bars across final ensemble model CV folds in logs
- Various UI improvements
- Various bug fixes

1.3.31 Version 1.0.23 (March 7, 2018)

- Support for Gains and Lift curves for binomial and multinomial classification
- Support for multi-GPU single-model training for large datasets
- Improved recipes for large datasets (faster and less memory/disk usage)
- Improved recipes for text features
- Increased sensitivity of interpretability setting for feature engineering complexity
- Disable automatic time column detection by default to avoid confusion
- Automatic column type conversion for test and validation data, and during scoring
- Improved speed of MLI
- Improved feature importances for MLI on transformed features
- Added ability to download each MLI plot as a PNG file
- Added support for dropped columns and weight column to MLI stand-alone page
- Fix serialization of bytes objects larger than 4 GiB
- Fix failure to build scoring pipeline with ‘command not found’ error
- Various UI improvements
- Various bug fixes

1.3.32 Version 1.0.22 (Feb 23, 2018)

- Fix CPU-only mode
- Improved robustness of datatable CSV parser

1.3.33 Version 1.0.21 (Feb 21, 2018)

- Fix MLI GUI scaling issue on Mac
- Work-around segfault in truncated SVD scipy backend
- Various bug fixes
1.3.34 Version 1.0.20 (Feb 17, 2018)

- HDFS/S3/Excel data connectors
- LDAP/PAM/Kerberos authentication
- Automatic setting of default values for accuracy / time / interpretability
- Interpretability: per-observation and per-feature (signed) contributions to predicted values in scoring pipeline
- Interpretability setting now affects feature engineering complexity and final model complexity
- Standalone MLI scoring pipeline for Python
- Time setting of 1 now runs for only 1 iteration
- Early stopping of experiments if convergence is detected
- ROC curve display for binomial and multinomial classification, with confusion matrices and threshold/F1/MCC display
- Training/Validation/Test data shift detectors
- Added AUCPR scorer for multinomial classification
- Improved handling of imbalanced binary classification problems
- Configuration file for runtime limits such as cores/memory/harddrive (for admins)
- Various GUI improvements (ability to rename experiments, re-run experiments, logs)
- Various bug fixes

1.3.35 Version 1.0.19 (Jan 28, 2018)

- Fix hang during final ensemble (accuracy >= 5) for larger datasets
- Allow scoring of all models built in older versions (>= 1.0.13) in GUI
- More detailed progress messages in the GUI during experiments
- Fix scoring pipeline to only use relative paths
- Error bars in model summary are now +/- 1*stddev (instead of 2*stddev)
- Added RMSPE scorer (RMS Percentage Error)
- Added SMAPE scorer (Symmetric Mean Abs. Percentage Error)
- Added AUCPR scorer (Area under Precision-Recall Curve)
- Gracefully handle inf/-inf in data
- Various UI improvements
- Various bug fixes

1.3.36 Version 1.0.18 (Jan 24, 2018)

- Fix migration from version 1.0.15 and earlier
- Confirmation dialog for experiment abort and data/experiment deletion
- Various UI improvements
- Various AutoVis improvements
• Various bug fixes

1.3.37 Version 1.0.17 (Jan 23, 2018)

• Fix migration from version 1.0.15 and earlier (partial, for experiments only)
• Added model summary download from GUI
• Restructured and renamed logs archive, and add model summary to it
• Fix regression in AutoVis in 1.0.16 that led to slowdown
• Various bug fixes

1.3.38 Version 1.0.16 (Jan 22, 2018)

• Added support for validation dataset (optional, instead of internal validation on training data)
• Standard deviation estimates for model scores (+/- 1 std.dev.)
• Computation of all applicable scores for final models (in logs only for now)
• Standard deviation estimates for MLI reason codes (+/- 1 std.dev.) when running in stand-alone mode
• Added ability to abort MLI job
• Improved final ensemble performance
• Improved outlier visualization
• Updated H2O-3 to version 3.16.0.4
• More readable experiment names
• Various speedups
• Various bug fixes

1.3.39 Version 1.0.15 (Jan 11, 2018)

• Fix truncated per-experiment log file
• Various bug fixes

1.3.40 Version 1.0.14 (Jan 11, 2018)

• Improved performance

1.3.41 Version 1.0.13 (Jan 10, 2018)

• Improved estimate of generalization performance for final ensemble by removing leakage from target encoding
• Added API for re-fitting and applying feature engineering on new (potentially larger) data
• Remove access to pre-transformed datasets to avoid unintended leakage issues downstream
• Added mean absolute percentage error (MAPE) scorer
• Enforce monotonicity constraints for binary classification and regression models if interpretability >= 6
• Use squared Pearson correlation for $R^2$ metric (instead of coefficient of determination) to avoid negative values
• Separated http and tcp scoring pipeline examples
• Reduced size of h2oai_client wheel
• No longer require weight column for test data if it was provided for training data
• Improved accuracy of final modeling pipeline
• Include H2O-3 logs in downloadable logs.zip
• Updated H2O-3 to version 3.16.0.2
• Various bug fixes

1.3.42 Version 1.0.11 (Dec 12, 2017)
• Faster multi-GPU training, especially for small data
• Increase default amount of exploration of genetic algorithm for systems with fewer than 4 GPUs
• Improved accuracy of generalization performance estimate for models on small data (< 100k rows)
• Faster abort of experiment
• Improved final ensemble meta-learner
• More robust date parsing
• Various bug fixes

1.3.43 Version 1.0.10 (Dec 4, 2017)
• Tool tips and link to documentation in parameter settings screen
• Faster training for multi-class problems with > 5 classes
• Experiment summary displayed in GUI after experiment finishes
• Python Client Library downloadable from the GUI
• Speedup for Maxwell-based GPUs
• Support for multinomial AUC and Gini scorers
• Add MCC and F1 scorers for binomial and multinomial problems
• Faster abort of experiment
• Various bug fixes

1.3.44 Version 1.0.9 (Nov 29, 2017)
• Support for time column for causal train/validation splits in time-series datasets
• Automatic detection of the time column from temporal correlations in data
• MLI improvements, dedicated page, selection of datasets and models
• Improved final ensemble meta-learner
• Test set score now displayed in experiment listing
• Original response is preserved in exported datasets
• Various bug fixes

1.3.45 Version 1.0.8 (Nov 21, 2017)
• Various bug fixes

1.3.46 Version 1.0.7 (Nov 17, 2017)
• Sharing of GPUs between experiments - can run multiple experiments at the same time while sharing GPU resources
• Persistence of experiments and data - can stop and restart the application without loss of data
• Support for weight column for optional user-specified per-row observation weights
• Support for fold column for user-specified grouping of rows in train/validation splits
• Higher accuracy through model tuning
• Faster training - overall improvements and optimization in model training speed
• Separate log file for each experiment
• Ability to delete experiments and datasets from the GUI
• Improved accuracy for regression tasks with very large response values
• Faster test set scoring - Significant improvements in test set scoring in the GUI
• Various bug fixes

1.3.47 Version 1.0.5 (Oct 24, 2017)
• Only display scorers that are allowed
• Various bug fixes

1.3.48 Version 1.0.4 (Oct 19, 2017)
• Improved automatic type detection logic
• Improved final ensemble accuracy
• Various bug fixes

1.3.49 Version 1.0.3 (Oct 9, 2017)
• Various speedups
• Results are now reproducible
• Various bug fixes
1.3.50 Version 1.0.2 (Oct 5, 2017)

- Improved final ensemble accuracy
- Weight of Evidence features added
- Various bug fixes

1.3.51 Version 1.0.1 (Oct 4, 2017)

- Improved speed of final ensemble
- Various bug fixes

1.3.52 Version 1.0.0 (Sep 24, 2017)

- Initial stable release
WHY DRIVERLESS AI?

Over the last several years, machine learning has become an integral part of many organizations’ decision-making processes at various levels. With not enough data scientists to fill the increasing demand for data-driven business processes, H2O.ai offers Driverless AI, which automates several time consuming aspects of a typical data science workflow, including data visualization, feature engineering, predictive modeling, and model explanation.

H2O Driverless AI is a high-performance, GPU-enabled computing platform for automatic development and rapid deployment of state-of-the-art predictive analytics models. It reads tabular data from plain text sources and a variety of external data sources and automates data visualization and building predictive models. Driverless AI targets business applications such as loss-given-default, probability of default, customer churn, campaign response, fraud detection, anti-money-laundering, demand forecasting, and predictive asset maintenance models. (Or in machine learning parlance: common regression, binomial classification, and multinomial classification problems.)

Visit https://www.h2o.ai/driverless-ai/ to download your free 21-day evaluation copy.

How do you frame business problems in a data set for Driverless AI?

The data that is read into Driverless AI must contain one entity per row, like a customer, patient, piece of equipment, or financial transaction. That row must also contain information about what you will be trying to predict using similar data in the future, like whether that customer in the row of data used a promotion, whether that patient was readmitted to the hospital within thirty days of being released, whether that piece of equipment required maintenance, or whether that financial transaction was fraudulent. (In data science speak, Driverless AI requires “labeled” data.) Driverless AI runs through your data many, many times looking for interactions, insights, and business drivers of the phenomenon described by the provided dataset. Driverless AI can handle simple data quality problems, but it currently requires all data for a single predictive model to be in the same dataset, and that dataset must have already undergone standard ETL, cleaning, and normalization routines before being loaded into Driverless AI.

How do you use Driverless AI results to create commercial value?

Commercial value is generated by Driverless AI in a few ways.

- Driverless AI empowers data scientists or data analysts to work on projects faster and more efficiently by using automation and state-of-the-art computing power to accomplish tasks in just minutes or hours instead of the weeks or months that it can take humans.

- Like in many other industries, automation leads to standardization of business processes, enforces best practices, and eventually drives down the cost of delivering the final product – in this case a predictive model.

- Driverless AI makes deploying predictive models easy – typically a difficult step in the data science process. In large organizations, value from predictive modeling is typically realized when a predictive model is moved from a data analyst’s or data scientist’s development environment into a production deployment setting. In this setting, the model is running on live data and making quick and automatic decisions that make or save money. Driverless AI provides both Java- and Python-based technologies to make production deployment simpler.

Moreover, the system was designed with interpretability and transparency in mind. Every prediction made by a Driverless AI model can be explained to business users, so the system is viable even for regulated industries.
Below are some of the key features available in Driverless AI.

### 3.1 Flexibility of Data and Deployment

Driverless AI works across a variety of data sources including Hadoop HDFS, Amazon S3, and more. Driverless AI can be deployed everywhere including all clouds (Microsoft Azure, AWS, Google Cloud) and on premises on any system, but it is ideally suited for systems with GPUs, including IBM Power 9 with GPUs built in.

### 3.2 NVIDIA GPU Acceleration

Driverless AI is optimized to take advantage of GPU acceleration to achieve up to 40X speedups for automatic machine learning. It includes multi-GPU algorithms for XGBoost, GLM, K-Means, and more. GPUs allow for thousands of iterations of model features and optimizations.

### 3.3 Data Visualization

For datasets, Driverless AI can generate visualizations and creates data plots that are most relevant from a statistical perspective based on the most relevant data statistics in order to help users get a quick understanding of their data prior to starting the model building process. See visualize-datasets for more information.

### 3.4 Automatic Feature Engineering

Feature engineering is the secret weapon that advanced data scientists use to extract the most accurate results from algorithms. H2O Driverless AI employs a library of algorithms and feature transformations to automatically engineer new, high value features for a given dataset. See Driverless AI Transformations for more information.

### 3.5 Machine Learning Interpretability (MLI)

Driverless AI provides robust interpretability of machine learning models to explain modeling results in a human-readable format. In the MLI view, Driverless AI employs a host of different techniques and methodologies for interpreting and explaining the results of its models. A number of charts are generated automatically, including K-LIME, Shapley, Variable Importance, Decision Tree Surrogate, Partial Dependence, Individual Conditional Expectation, and
more. Additionally, you can download a CSV of LIME and Shapley reasons codes from this view. See Interpreting a Model for more information.

3.6 Time Series

Driverless AI delivers superior time series capabilities to optimize for almost any prediction time window. Driverless AI incorporates data from numerous predictors, handles structured character data and high-cardinality categorical variables, and handles gaps in time series data and other missing values. See Time Series in Driverless AI for more information.

3.7 NLP with TensorFlow

Text data can contain critical information to inform better predictions. Driverless AI automatically converts short text strings into features using powerful techniques like TFIDF. With TensorFlow, Driverless AI can also process larger text blocks and build models using all available data to solve business problems like sentiment analysis, document classification, and content tagging. See NLP in Driverless AI for more information.

3.8 Automatic Scoring Pipelines

For completed experiments, Driverless AI automatically generates both Python scoring pipelines and new ultra-low latency automatic scoring pipelines. The new automatic scoring pipeline is a unique technology that deploys all feature engineering and the winning machine learning model in a highly optimized, low-latency, production-ready Java code that can be deployed anywhere. See The Python and MOJO Scoring Pipelines for more information.
4.1 XGBoost

XGBoost is a supervised learning algorithm that implements a process called boosting to yield accurate models. Boosting refers to the ensemble learning technique of building many models sequentially, with each new model attempting to correct for the deficiencies in the previous model. In tree boosting, each new model that is added to the ensemble is a decision tree. XGBoost provides parallel tree boosting (also known as GBDT, GBM) that solves many data science problems in a fast and accurate way. For many problems, XGBoost is one of the best gradient boosting machine (GBM) frameworks today.

4.2 LightGBM

LightGBM is a gradient boosting framework developed by Microsoft that uses tree based learning algorithms. It was specifically designed for lower memory usage and faster training speed and higher efficiency. Similar to XGBoost, it is one of the best gradient boosting implementations available. It is also used for fitting Random Forest models inside of Driverless AI.

Note: LightGBM with GPUs is not supported on Power currently.

4.3 GLM

Generalized Linear Models (GLM) estimate regression models for outcomes following exponential distributions. GLMs are an extension of traditional linear models. They have gained popularity in statistical data analysis due to:

- the flexibility of the model structure unifying the typical regression methods (such as linear regression and logistic regression for binary classification)
- the recent availability of model-fitting software
- the ability to scale well with large datasets

4.4 TensorFlow

TensorFlow is an open source software library for performing high performance numerical computation. Driverless AI includes a TensorFlow NLP recipe based on CNN Deep learning models.

Note: MOJOs are currently not available for TensorFlow models.
4.5 RuleFit

The RuleFit [1] algorithm creates an optimal set of decision rules by first fitting a tree model, and then fitting a Lasso (L1-regularized) GLM model to create a linear model consisting of the most important tree leaves (rules).

Note: MOJOs are currently not available for RuleFit models.

4.6 FTRL

Follow the Regularized Leader (FTRL) is a DataTable implementation [2] of the FTRL-Proximal online learning algorithm proposed in [3]. This implementation uses a hashing trick and Hogwild approach [4] for parallelization. FTRL supports binomial and multinomial classification for categorical targets, as well as regression for continuous targets.

Note: MOJOs are currently not available for FTRL models.

4.7 References


CHAPTER
FIVE

INSTALLING AND UPGRADING DRIVERLESS AI

For the best (and intended-as-designed) experience, install Driverless AI on modern data center hardware with GPUs and CUDA support. Use Pascal or Volta GPUs with maximum GPU memory for best results. (Note that the older K80 and M60 GPUs available in EC2 are supported and very convenient, but not as fast.)

Driverless AI supports local, LDAP, and PAM authentication. Authentication can be configured by setting environment variables or via a config.toml file. Refer to the Configuring Authentication section for more information. Note that the default authentication method is “unvalidated.”

Driverless AI also supports HDFS, S3, Google Cloud Storage, Google Big Query, KDB, Minio, and Snowflake access. Support for these data sources can be configured by setting environment variables for the data connectors or via a config.toml file. Refer to the Enabling Data Connectors section for more information.

5.1 Before You Begin the Installation

Please review the following information before you begin installing Driverless AI. Be sure to also review the Sizing Requirements in the next section before beginning the installation.

5.1.1 Note about nvidia-docker 1.0

If you have nvidia-docker 1.0 installed, you need to remove it and all existing GPU containers. Refer to https://github.com/NVIDIA/nvidia-docker/blob/master/README.md for more information.

5.1.2 Note about CUDA versions

Your host environment must have CUDA 9.0 or later drivers installed. Driverless AI ships with it’s own CUDA libraries, but the driver must exist in the host environment.

5.1.3 Note about Authentication

The default authentication setting in Driverless AI is “unvalidated.” In this case, Driverless AI will accept any login and password combination, it will not validate whether the password is correct for the specified login ID, and it will connect to the system as the user specified in the login ID. This is true for all instances, including Cloud, Docker, and native instances.

We recommend that you configure authentication. Driverless AI provides a number of authentication options, including LDAP, PAM, Local, and None. Refer to Configuring Authentication for information on how to enable a different authentication method.
Note: Driverless AI is also integrated with IBM Spectrum Conductor and supports authentication from Conductor. Contact sales@h2o.ai for more information about using IBM Spectrum Conductor authentication.

5.2 Sizing Requirements

5.2.1 Sizing Requirements for Native Installs

Driverless AI requires a minimum of 5 GB of system memory in order to start experiments and a minimum of 5 GB of disk space in order to run a small experiment. Note that these limits can changed in the config.toml file. We recommend that you have lots of system CPU memory (64 GB or more) and 1 TB of free disk space available.

5.2.2 Sizing Requirements for Docker Installs

For Docker installs, we recommend 1 TB of free disk space. Driverless AI uses approximately 38 GB. In addition, the unpacking/temp files require space on the same Linux mount /var during installation. Once DAI runs, the mounts from the Docker container can point to other file system mount points.

5.2.3 GPU Sizing Requirements

If you are running Driverless AI with GPUs, be sure that your GPU has compute capability >=3.5 and at least 4GB of RAM. If these requirements are not met, then Driverless AI will switch to CPU-only mode.

5.2.4 Sizing Requirements for Storing Experiments

We recommend that your tmp directory has at least 500 GB to 1 TB of space. The tmp directory holds all experiments and all datasets. We also recommend that you use SSDs (preferably NVMe).

5.2.5 Virtual Memory Settings in Linux

If you are running Driverless AI on a Linux machine, we recommend setting the overcommit memory to 0. The setting can be changed by the following command:

```
sudo echo 0 > /proc/sys/vm/overcommit_memory
```

This is the default value, and it indicates that the Linux kernel is free to overcommit memory. If this value is set to 2, then the Linux kernel will not overcommit memory. In this case, the memory requirements of Driverless AI may surpass the memory allocation limit, which would prevent the experiment from completing.

5.3 Linux X86_64 Installs

This section provides installation steps for Linux 86_64 environments. This includes information for Docker image installs, RPMs, Deb, and Tar installs as well as Cloud installations.
5.3.1 Linux Docker Images

To simplify local installation, Driverless AI is provided as a Docker image for the following system combinations:

<table>
<thead>
<tr>
<th>Host OS</th>
<th>Docker Version</th>
<th>Host Architecture</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu 16.04 or later</td>
<td>Docker CE</td>
<td>x86_64</td>
<td>64 GB</td>
</tr>
<tr>
<td>RHEL or CentOS 7.4 or later</td>
<td>Docker CE</td>
<td>x86_64</td>
<td>64 GB</td>
</tr>
<tr>
<td>NVIDIA DGX Registry</td>
<td></td>
<td>x86_64</td>
<td></td>
</tr>
</tbody>
</table>

**Note:** Cuda9.0 or later with NVIDIA drivers >= 396.

For the best performance, including GPU support, use nvidia-docker. For a lower-performance experience without GPUs, use regular docker (with the same docker image).

These installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit https://www.h2o.ai/driverless-ai/. Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

**Install on Ubuntu**

This section describes how to install the Driverless AI Docker image on Ubuntu. The installation steps vary depending on whether your system has GPUs or if it is CPU only.

**Environment**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>GPUs?</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu with GPUs</td>
<td>Yes</td>
<td>64 GB</td>
</tr>
<tr>
<td>Ubuntu with CPUs</td>
<td>No</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

**Install on Ubuntu with GPUs**

**Note:** Driverless AI is supported on Ubuntu 16.04 or later.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/. (Note that the contents of this Docker image include a CentOS kernel and CentOS packages.)

2. Install and run Docker on Ubuntu (if not already installed):

   ```bash
   # Install and run Docker on Ubuntu
   curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
   sudo apt-key fingerprint 0EBFCD88 sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -
   ->cs) stable"
   sudo apt-get update
   sudo apt-get install docker-ce
   sudo systemctl start docker
   ```

curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \n  sudo apt-key add -
distribution=$(. /etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
  docker.list | \n  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update

# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2

4. Verify that the NVIDIA driver is up and running. If the driver is not up and running, log on to http://www.nvidia.com/Download/index.aspx?lang=en-us to get the latest NVIDIA Tesla V/P/K series driver:

nvidia-smi

5. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

# Set up directory with the version name
mkdir dai_rel_VERSION

6. Change directories to the new folder, then load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI. Replace VERSION with your image.

# cd into the new directory
cd dai_rel_VERSION

# Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz

7. Enable persistence of the GPU. Note that this needs to be run once every reboot. Refer to the following for more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.

nvidia-persistenced --user <USER>
nvidia-smi -pm 1

8. Set up the data, log, and license directories on the host machine:

# Set up the data, log, license, and tmp directories on the host machine.
  (within the new directory)
mkdir data
mkdir log
mkdir license
mkdir tmp

9. At this point, you can copy data into the data directory on the host machine. The data will be visible inside the Docker container.

10. Run docker images to find the image tag.

11. Start the Driverless AI Docker image with nvidia-docker and replace TAG below with the image tag:

# Start the Driverless AI Docker image
nvidia-docker run \
  --pid=host \
  --init \

Using Driverless AI, Release 1.6.2

Driverless AI will begin running:

```
Welcome to H2O.ai's Driverless AI
--------------------------------
- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```

12. Connect to Driverless AI with your browser:

```
```

Install on Ubuntu with CPUs

Note: Driverless AI is supported on Ubuntu 16.04 or later.

This section describes how to install and start the Driverless AI Docker image on Ubuntu. Note that this uses Docker EE and not NVIDIA Docker. GPU support will not be available.

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.

2. Install and run Docker on Ubuntu (if not already installed):

```
# Install and run Docker on Ubuntu
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88 sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
sudo apt-get update
sudo apt-get install docker-ce
sudo systemctl start docker
```

3. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

```
# Set up directory with the version name
mkdir dai_rel_VERSION
```
4. Change directories to the new folder, then load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI. Replace VERSION with your image.

```bash
# cd into the new directory
cd dai_rel_VERSION

# Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

5. Set up the data, log, license, and tmp directories on the host machine (within the new directory):

```bash
# Set up the data, log, license, and tmp directories
mkdir data
mkdir log
mkdir license
mkdir tmp
```

6. At this point, you can copy data into the data directory on the host machine. The data will be visible inside the Docker container.

7. Run `docker images` to find the new image tag.

8. Start the Driverless AI Docker image and replace TAG below with the image tag. Note that GPU support will not be available.

```bash
# Start the Driverless AI Docker image
docker run \
  --pid=host \
  --init \
  --rm \
  --shm-size=256m \
  -u `id -u`:`id -g` \
  -p 12345:12345 \
  -v `pwd`/data:/data \
  -v `pwd`/log:/log \
  -v `pwd`/license:/license \
  -v `pwd`/tmp:/tmp \
  h2oai/dai-centos7-x86_64:TAG
```

Driverless AI will begin running:

```
Welcome to H2O.ai's Driverless AI
----------------------------------
- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```

9. Connect to Driverless AI with your browser:

Stopping the Docker Image

To stop the Driverless AI Docker image, type Ctrl + C in the Terminal (Mac OS X) or PowerShell (Windows 10) window that is running the Driverless AI Docker image.

Upgrading the Docker Image

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

WARNING: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

Note: Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:
   ```
   # Set up directory with the version name
   mkdir dai_rel_VERSION
   # cd into the new directory
   cd dai_rel_VERSION
   ```
3. Retrieve the Driverless AI package from https://www.h2o.ai/download/ and add it to the new directory.
4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace VERSION with your image.
   ```
   # Load the Driverless AI docker image
   docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
   ```
5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:
   ```
   # Copy the data, log, license, and tmp directories on the host machine
   cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
   cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
   cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
   cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
   ```
   At this point, your experiments from the previous versions will be visible inside the Docker container.
6. Use docker images to find the new image tag.
7. Start the Driverless AI Docker image.

Install on RHEL

This section describes how to install the Driverless AI Docker image on RHEL. The installation steps vary depending on whether your system has GPUs or if it is CPU only.

Environment

<table>
<thead>
<tr>
<th>Operating System</th>
<th>GPUs?</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL with GPUs</td>
<td>Yes</td>
<td>64 GB</td>
</tr>
<tr>
<td>RHEL with CPUs</td>
<td>No</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

Install on RHEL with GPUs

Note: Refer to the following links for more information about using RHEL with GPUs. These links describe how to disable automatic updates and specific package updates. This is necessary in order to prevent a mismatch between the NVIDIA driver and the kernel, which can lead to the GPUs failures.

- https://access.redhat.com/solutions/2372971

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

Note: As of this writing, Driverless AI has only been tested on RHEL version 7.4.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.
2. Install and start Docker EE on RHEL (if not already installed). Follow the instructions on https://docs.docker.com/engine/installation/linux/docker-ee/rhel/.

   Alternatively, you can run on Docker CE.

   ```
   sudo yum install --enablerepo=epel,yum-utils
   sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/
   docker-ce.repo
   sudo yum makecache fast
   sudo yum -y install docker-ce
   sudo systemctl start docker
   ```


   ```
   curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \\
   sudo apt-key add -
   distribution=$(/etc/os-release;echo $ID$VERSION_ID)
   curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
   docker.list | \\
   ```

   (continues on next page)
Using Driverless AI, Release 1.6.2

(continued from previous page)

```
sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update

# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2
```

**Note:** If you would like the nvidia-docker service to automatically start when the server is rebooted then run the following command. If you do not run this commend, you will have to remember to start the nvidia-docker service manually; otherwise the GPUs will not appear as available.

```
sudo systemctl enable nvidia-docker
```

Alternatively, if you have installed Docker CE above you can install nvidia-docker with:

```
curl -s -L https://nvidia.github.io/nvidia-docker/centos7/x86_64/nvidia-
→docker.repo | \
sudo tee /etc/yum.repos.d/nvidia-docker.repo
sudo yum install nvidia-docker2
```

4. Verify that the NVIDIA driver is up and running. If the driver is not up and running, log on to http://www.nvidia.com/Download/index.aspx?lang=en-us to get the latest NVIDIA Tesla V/P/K series driver.

```
nvidia-docker run --rm nvidia/cuda nvidia-smi
```

5. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

```
# Set up directory with the version name
mkdir dai_rel_VERSION
```

6. Change directories to the new folder, then load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI. Replace VERSION with your image.

```
# cd into the new directory
cd dai_rel_VERSION

# Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

7. Enable persistence of the GPU. Note that this needs to be run once every reboot. Refer to the following for more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.

```
nvidia-persistenced --user <USER>
nvidia-smi -pm 1
```

8. Set up the data, log, and license directories on the host machine (within the new directory):

```
# Set up the data, log, license, and tmp directories on the host machine
mkdir data
mkdir log
mkdir license
mkdir tmp
```

9. At this point, you can copy data into the data directory on the host machine. The data will be visible inside the Docker container.

10. Run `docker images` to find the image tag.

5.3. Linux X86_64 Installs
11. Start the Driverless AI Docker image with nvidia-docker and replace TAG below with the image tag:

```
# Start the Driverless AI Docker image
nvidia-docker run \
    --pid=host \
    --init \
    --rm \
    --shm-size=256m \
    -u `id -u`:`id -g` \
    -p 12345:12345 \
    -v `pwd`/data:/data \
    -v `pwd`/log:/log \
    -v `pwd`/license:/license \
    -v `pwd`/tmp:/tmp \
    h2oai/dai-centos7-x86_64:TAG
```

Driverless AI will begin running:

```
Welcome to H2O.ai's Driverless AI

- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```


### Install on RHEL with CPUs

This section describes how to install and start the Driverless AI Docker image on RHEL. Note that this uses Docker EE and not NVIDIA Docker. GPU support will not be available.

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

Note: As of this writing, Driverless AI has only been tested on RHEL version 7.4.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Install and start Docker EE on RHEL (if not already installed). Follow the instructions on https://docs.docker.com/engine/installation/linux/docker-ee/rhel/.

   Alternatively, you can run on Docker CE.

   ```
   sudo yum install -y yum-utils
   sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/\n   →docker-ce.repo
   sudo yum makecache fast
   sudo yum -y install docker-ce
   sudo systemctl start docker
   ```

2. On the machine that is running Docker EE, retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.

3. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:
4. Load the Driverless AI Docker image inside the new directory. The following example shows how to load Driverless AI version. Replace VERSION with your image.

```bash
# Load the Driverless AI Docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

5. Set up the data, log, license, and tmp directories (within the new directory):

```bash
# Set up the directory associated with the selected version of Driverless AI
cd dai_rel_VERSION

# Set up the data, log, license, and tmp directories on the host machine
mkdir data
mkdir log
mkdir license
mkdir tmp
```

6. Copy data into the `data` directory on the host. The data will be visible inside the Docker container at `/<user-home>/data`.

7. Run `docker images` to find the image tag.

8. Start the Driverless AI Docker image and replace TAG below with the image tag. Note that GPU support will not be available.

```bash
$ docker run \
  --pid=host \
  --init \
  --rm \
  --shm-size=256m \
  -u `id -u`:`id -g` \
  -p 12345:12345 \
  -v `pwd`/data:/data \
  -v `pwd`/log:/log \
  -v `pwd`/license:/license \
  -v `pwd`/tmp:/tmp \
  h2oai/dai-centos7-x86_64:TAG
```

Driverless AI will begin running:

```
Welcome to H2O.ai's Driverless AI

- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```


5.3. Linux X86_64 Installs 39
Stopping the Docker Image

To stop the Driverless AI Docker image, type `Ctrl + C` in the Terminal (Mac OS X) or PowerShell (Windows 10) window that is running the Driverless AI Docker image.

Upgrading the Docker Image

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

**Note:** Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:

   ```bash
   # Set up directory with the version name
   mkdir dai_rel_VERSION
   # cd into the new directory
   cd dai_rel_VERSION
   ```

3. Retrieve the Driverless AI package from [https://www.h2o.ai/download/](https://www.h2o.ai/download/) and add it to the new directory.

4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace `VERSION` with your image.

   ```bash
   # Load the Driverless AI docker image
   docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
   ```

5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:

   ```bash
   # Copy the data, log, license, and tmp directories on the host machine
   cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
   cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
   cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
   cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
   ```

   At this point, your experiments from the previous versions will be visible inside the Docker container.

6. Use `docker images` to find the new image tag.
7. Start the Driverless AI Docker image.


Install on NVIDIA GPU Cloud/NGC Registry

Driverless AI is supported on the following NVIDIA DGX products, and the installation steps for each platform are the same.

- NVIDIA GPU Cloud
- NVIDIA DGX-1
- NVIDIA DGX-2
- NVIDIA DGX Station

<table>
<thead>
<tr>
<th>Provider</th>
<th>GPUs</th>
<th>Min Memory</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVIDIA GPU Cloud</td>
<td>Yes</td>
<td></td>
<td>Serious use</td>
</tr>
<tr>
<td>NVIDIA DGX-1/DGX-2</td>
<td>Yes</td>
<td>128 GB</td>
<td>Serious use</td>
</tr>
<tr>
<td>NVIDIA DGX Station</td>
<td>Yes</td>
<td>64 GB</td>
<td>Serious Use</td>
</tr>
</tbody>
</table>

Installing the NVIDIA NGC Registry

Note: These installation instructions assume that you are running on an NVIDIA DGX machine. Driverless AI is only available in the NGC registry for DGX machines.

1. Log in to your NVIDIA GPU Cloud account at https://ngc.nvidia.com/registry. (Note that NVIDIA Compute is no longer supported by NVIDIA.)

2. In the Registry > Partners menu, select h2oai-driverless.
3. At the bottom of the screen, select one of the H2O Driverless AI tags to retrieve the pull command.

4. On your NVIDIA DGX machine, open a command prompt and use the specified pull command to retrieve the Driverless AI image. For example:

```bash
docker pull nvcr.io/nvidia_partners/h2o-driverless-ai:latest
```

5. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

```bash
# Set up directory with the version name
mkdir dai_rel_VERSION
```

6. Set up the data, log, license, and tmp directories on the host machine:

```bash
# cd into the directory associated with the selected version of Driverless AI
cd dai_rel_VERSION

# Set up the data, log, license, and tmp directories on the host machine
mkdir data
mkdir log
mkdir license
mkdir tmp
```

7. At this point, you can copy data into the data directory on the host machine. The data will be visible inside the Docker container.

8. Enable persistence of the GPU. Note that this only needs to be run once. Refer to the following for more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.

```bash
nvidia-persistenced --user <USER>
nvidia-smi -pm 1
```

9. Run `docker images` to find the new image tag.

10. Start the Driverless AI Docker image with nvidia-docker and replace TAG below with the image tag:
nvidia-docker run \
  --pid=host \n  --init \n  --rm \n  --shm-size=256m \n  -u `id -u`:`id -g` \n  -p 12345:12345 \n  -v `pwd`/data:/data \n  -v `pwd`/log:/log \n  -v `pwd`/license:/license \n  -v `pwd`/tmp:/tmp \n  nvcr.io/h2oai/h2oai-driverless-ai:TAG

Driverless AI will begin running:

--------------------------------
Welcome to H2O.ai's Driverless AI
--------------------------------
- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container

11. Connect to Driverless AI with your browser:

Stopping Driverless AI

Use Ctrl+C to stop Driverless AI.

Upgrading Driverless AI

The steps for upgrading Driverless AI on an NVIDIA DGX system are similar to the installation steps.

WARNING: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

Note: Use Ctrl+C to stop Driverless AI if it is still running.

1. On your NVIDIA DGX machine, create a directory for the new Driverless AI version.

5.3. Linux X86_64 Installs
2. Copy the data, log, license, and tmp directories from the previous Driverless AI directory into the new Driverless AI directory.

3. Run `docker pull nvcr.io/h2oai/h2oai-driverless-ai:latest` to retrieve the latest Driverless AI version.

4. Start the Driverless AI Docker image.


### 5.3.2 Linux RPMs

For Linux machines that will not use the Docker image or DEB, an RPM installation is available for the following environments:

- x86_64 RHEL 7, CentOS 7, or SLES 12

The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/products/h2o-driverless-ai/](https://www.h2o.ai/products/h2o-driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

#### Environment

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL with GPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>RHEL with CPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>CentOS 7 with GPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>CentOS 7 with CPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>SLES 12 with GPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>SLES 12 with CPUs</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

#### Requirements

- RedHat 7/CentOS 7/SLES 12
- Cuda 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- OpenCL (Required for LightGBM support on GPUs.)
- Driverless AI RPM, available from [https://www.h2o.ai/download/](https://www.h2o.ai/download/)

#### About the Install

- The ‘dai’ service user is created locally (in `/etc/passwd`) if it is not found by ‘getent passwd’. You can override the user by providing the DAI_USER environment variable during rpm or dpkg installation.
- The ‘dai’ service group is created locally (in `/etc/group`) if it is not found by ‘getent group’. You can override the group by providing the DAI_GROUP environment variable during rpm or dpkg installation.
- Configuration files are put in `/etc/dai` and owned by the ‘root’ user:
  - `/etc/dai/config.toml`: Driverless AI config file (See Using the config.toml File section for details)
Using Driverless AI, Release 1.6.2

- /etc/dai/User.conf: Systemd config file specifying the service user
- /etc/dai/Group.conf: Systemd config file specifying the service group
- /etc/dai/EnvironmentFile.conf: Systemd config file specifying (optional) environment variable overrides

- Software files are put in /opt/h2oai/dai and owned by the ‘root’ user
- The following directories are owned by the service user so they can be updated by the running software:
  - /opt/h2oai/dai/home: The application’s home directory (license key files are stored here)
  - /opt/h2oai/dai/tmp: Experiments and imported data are stored here
  - /opt/h2oai/dai/log: Log files go here if you are not using systemd (if you are using systemd, then use the standard journalctl tool)

- By default, Driverless AI looks for a license key in /opt/h2oai/dai/home/driverlessai/license.sig. If you are installing Driverless AI programmatically, you can copy a license key file to that location. If no license key is found, the application will interactively guide you to add one from the Web UI.
- systemd unit files are put in /usr/lib/systemd/system
- Symbolic links to the configuration files in /etc/dai files are put in /etc/systemd/system

If your environment is running an operational systemd, that is the preferred way to manage Driverless AI. The package installs the following systemd services and a wrapper service:

- dai: Wrapper service that starts/stops the other three services
- dai-dai: Main Driverless AI process
- dai-h2o: H2O-3 helper process used by Driverless AI
- dai-procsy: Procsy helper process used by Driverless AI
- dai-vis-server: Visualization server helper process used by Driverless AI

If you don’t have systemd, you can also use the provided run script to start Driverless AI.

**Installing Driverless AI**

Run the following commands to install the Driverless AI RPM. Replace VERSION with your specific version.

```
# Install Driverless AI.
sudo rpm -i dai-VERSION.rpm
```

**Note:** For RHEL 7.5, it is necessary to upgrade library glib2:

```
sudo yum upgrade glib2
```

By default, the Driverless AI processes are owned by the ‘dai’ user and ‘dai’ group. You can optionally specify a different service user and group as shown below. Replace <myuser> and <mygroup> as appropriate.

```
# Temporarily specify service user and group when installing Driverless AI.
# rpm saves these for systemd in the /etc/dai/User.conf and /etc/dai/Group.conf files.
sudo DAI_USER=myuser DAI_GROUP=mygroup rpm -i dai-VERSION.rpm
```

You may now optionally make changes to /etc/dai/config.toml.
Starting Driverless AI

If you have systemd (preferred):

```bash
# Start Driverless AI.
sudo systemctl start dai
```

If you do not have systemd:

```bash
# Start Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

Starting NVIDIA Persistence Mode

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.

```bash
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```

Install OpenCL

OpenCL is required in order to run LightGBM on GPUs. Run the following for Centos7/RH7 based systems using yum and x86.

```bash
yum -y clean all
yum -y makecache
yum -y update
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/c/clinfo-2.1.17.02.09-1.el7.x86_64.rpm
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/o/ocl-icd-2.2.12-1.el7.x86_64.rpm
rpm -i clinfo-2.1.17.02.09-1.el7.x86_64.rpm
rpm -i ocl-icd-2.2.12-1.el7.x86_64.rpm
clinfo

mkdir -p /etc/OpenCL/vendors && 
    echo "libnvidia-openc1.so.1" > /etc/OpenCL/vendors/nvidia.icd
```

Looking at Driverless AI log files

If you have systemd (preferred):

```bash
sudo systemctl status dai-dai
sudo systemctl status dai-h2o
sudo systemctl status dai-procsy
sudo systemctl status dai-vis-server
sudo journalctl -u dai-dai
sudo journalctl -u dai-h2o
sudo journalctl -u dai-procsy
sudo journalctl -u dai-vis-server
```

If you do not have systemd:
Using Driverless AI, Release 1.6.2

```bash
sudo less /opt/h2oai/dai/log/dai.log
sudo less /opt/h2oai/dai/log/h2o.log
sudo less /opt/h2oai/dai/log/procsy.log
sudo less /opt/h2oai/dai/log/vis-server.log
```

**Stopping Driverless AI**

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai
```

**Upgrading Driverless AI**

**WARNING**: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

The upgrade process inherits the service user and group from `/etc/dai/User.conf` and `/etc/dai/Group.conf`. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time.
```

(continues on next page)
Upgrade and restart.

```
sudo rpm -U dai-NEWVERSION.rpm
sudo systemctl daemon-reload
sudo systemctl start dai
```

If you do not have systemd:

```
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade and restart.
sudo rpm -U dai-NEWVERSION.rpm
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

Uninstalling Driverless AI

If you have systemd (preferred):

```
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall.
sudo rpm -e dai
```

If you do not have systemd:

```
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall.
sudo rpm -e dai
```

**CAUTION!** At this point you can optionally completely remove all remaining files, including the database. (This cannot be undone.)

```
sudo rm -rf /opt/h2oai/dai
sudo rm -rf /etc/dai
```

### 5.3.3 Linux DEBs

For Linux machines that will not use the Docker image or RPM, a DEB installation is available for x86_64 Ubuntu 16.04/18.04.
The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit https://www.h2o.ai/products/h2o-driverless-ai/. Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

**Environment**

<table>
<thead>
<tr>
<th>Operating System</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ubuntu with GPUs</td>
<td>64 GB</td>
</tr>
<tr>
<td>Ubuntu with CPUs</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

**Requirements**

- Ubuntu 16.04/Ubuntu 18.04
- Cuda 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- OpenCL (Required for LightGBM support on GPUs.)
- Driverless AI DEB, available from https://www.h2o.ai/download/

**About the Install**

- The ‘dai’ service user is created locally (in /etc/passwd) if it is not found by ‘getent passwd’. You can override the user by providing the DAI_USER environment variable during rpm or dpkg installation.
- The ‘dai’ service group is created locally (in /etc/group) if it is not found by ‘getent group’. You can override the group by providing the DAI_GROUP environment variable during rpm or dpkg installation.
- Configuration files are put in /etc/dai and owned by the ‘root’ user:
  - /etc/dai/config.toml: Driverless AI config file (See *Using the config.toml File* section for details)
  - /etc/dai/User.conf: Systemd config file specifying the service user
  - /etc/dai/Group.conf: Systemd config file specifying the service group
  - /etc/dai/EnvironmentFile.conf: Systemd config file specifying (optional) environment variable overrides
- Software files are put in /opt/h2oai/dai and owned by the ‘root’ user
- The following directories are owned by the service user so they can be updated by the running software:
  - /opt/h2oai/dai/home: The application’s home directory (license key files are stored here)
  - /opt/h2oai/dai/tmp: Experiments and imported data are stored here
  - /opt/h2oai/dai/log: Log files go here if you are not using systemd (if you are using systemd, then the use the standard journalctl tool)
- By default, Driverless AI looks for a license key in /opt/h2oai/dai/home/driverlessai/license.sig. If you are installing Driverless AI programmatically, you can copy a license key file to that location. If no license key is found, the application will interactively guide you to add one from the Web UI.
- systemd unit files are put in /usr/lib/systemd/system
- Symbolic links to the configuration files in /etc/dai files are put in /etc/systemd/system
Using Driverless AI, Release 1.6.2

If your environment is running an operational systemd, that is the preferred way to manage Driverless AI. The package installs the following systemd services and a wrapper service:

- **dai**: Wrapper service that starts/stops the other three services
- **dai-dai**: Main Driverless AI process
- **dai-h2o**: H2O-3 helper process used by Driverless AI
- **dai-procsy**: Procsy helper process used by Driverless AI
- **dai-vis-server**: Visualization server helper process used by Driverless AI

If you don’t have systemd, you can also use the provided run script to start Driverless AI.

### Starting NVIDIA Persistence Mode (GPU only)

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: [http://docs.nvidia.com/deploy/driver-persistence/index.html](http://docs.nvidia.com/deploy/driver-persistence/index.html).

```
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```

### Install OpenCL

OpenCL is required in order to run LightGBM on GPUs. Run the following for Ubuntu-based systems.

```
sudo apt-get install opencl-headers clinfo ocl-icd-opencl-dev

mkdir -p /etc/OpenCL/vendors && \
  echo "libnvidia-opencl.so.1" > /etc/OpenCL/vendors/nvidia.icd
```

### Installing the Driverless AI Linux DEB

Run the following commands to install the Driverless AI DEB. Replace VERSION with your specific version.

```
# Install Driverless AI.
sudo dpkg -i dai_VERSION.deb
```

By default, the Driverless AI processes are owned by the ‘dai’ user and ‘dai’ group. You can optionally specify a different service user and group as shown below. Replace <myuser> and <mygroup> as appropriate.

```
# Temporarily specify service user and group when installing Driverless AI.
# dpkg saves these for systemd in the /etc/dai/User.conf and /etc/dai/Group.conf_files.
sudo DAI_USER=myuser DAI_GROUP=mygroup dpkg -i dai_VERSION.deb
```

You may now optionally make changes to `/etc/dai/config.toml`.

### Starting Driverless AI

If you have systemd (preferred):

```
# Start Driverless AI.
sudo systemctl start dai
```
If you do not have systemd:

```bash
# Start Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

### Looking at Driverless AI log files

If you have systemd (preferred):

```bash
sudo systemctl status dai-dai
sudo systemctl status dai-h2o
sudo systemctl status dai-procsy
sudo systemctl status dai-vis-server
sudo journalctl -u dai-dai
sudo journalctl -u dai-h2o
sudo journalctl -u dai-procsy
sudo journalctl -u dai-vis-server
```

If you do not have systemd:

```bash
sudo less /opt/h2oai/dai/log/dai.log
sudo less /opt/h2oai/dai/log/h2o.log
sudo less /opt/h2oai/dai/log/procsy.log
sudo less /opt/h2oai/dai/log/vis-server.log
```

### Stopping Driverless AI

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai
```

# The processes should now be stopped. Verify.
```bash
sudo ps -u dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai
```

# The processes should now be stopped. Verify.
```bash
sudo ps -u dai
```

### Upgrading Driverless AI

**WARNING**: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want
Using Driverless AI, Release 1.6.2

to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

The upgrade process inherits the service user and group from /etc/dai/User.conf and /etc/dai/Group.conf. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade Driverless AI.
sudo dpkg -i dai_NEWVERSION.deb
sudo systemctl daemon-reload
sudo systemctl start dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time. If you do not, all previous data will be lost.

# Upgrade and restart.
sudo dpkg -i dai_NEWVERSION.deb
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

Uninstalling Driverless AI

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall Driverless AI.
sudo dpkg -r dai

# Purge Driverless AI.
sudo dpkg -P dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai
```

(continues on next page)
Using Driverless AI, Release 1.6.2

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall Driverless AI.
sudo dpkg -r dai

# Purge Driverless AI.
sudo dpkg -P dai

CAUTION! At this point you can optionally completely remove all remaining files, including the database (this cannot be undone):

```
sudo rm -rf /opt/h2oai/dai
sudo rm -rf /etc/dai
```

**Common Problems**

**Start of Driverless AI fails on the message “Segmentation fault (core dumped)” on Ubuntu 18.**

This problem is caused by the font NotoColorEmoji.ttf, which cannot be processed by the Python matplotlib library. A workaround is to disable the font by renaming it. (Do not use fontconfig because it is ignored by matplotlib.) The following will print out the command that should be executed.

```
sudo find / -name "NotoColorEmoji.ttf" 2>/dev/null | xargs -I{} echo sudo mv {} {}.
```

5.3.4 Linux TAR SH

The Driverless AI software is available for use in pure user-mode environments as a self-extracting TAR SH archive. This form of installation does not require a privileged user to install or to run.

This artifact has the same compatibility matrix as the RPM and DEB packages (combined), it just comes packaged slightly differently. See those sections for a full list of supported environments.

The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/products/h2o-driverless-ai/](https://www.h2o.ai/products/h2o-driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in.

**Requirements**

- RedHat 7 or Ubuntu 16.04
- CUDA 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- OpenCL (Required for LightGBM support on GPUs.)
- Driverless AI TAR SH, available from [https://www.h2o.ai/download/](https://www.h2o.ai/download/)

5.3. Linux X86_64 Installs
Installing Driverless AI

Run the following commands to install the Driverless AI RPM. Replace VERSION with your specific version.

```
# Install Driverless AI.
chmod 755 dai-VERSION.sh
./dai-VERSION.sh
```

You may now cd to the unpacked directory and optionally make changes to `config.toml`.

Starting Driverless AI

```
# Start Driverless AI.
./run-dai.sh
```

Starting NVIDIA Persistence Mode

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: [http://docs.nvidia.com/deploy/driver-persistence/index.html](http://docs.nvidia.com/deploy/driver-persistence/index.html).

```
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```

Install OpenCL

OpenCL is required in order to run LightGBM on GPUs. Run the following for Centos7/RH7 based systems using yum and x86.

```
yum -y clean all
yum -y makecache
yum -y update
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/c/clinfo-2.1.17.02.09-1.el7.x86_64.rpm
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/o/ocl-icd-2.2.12-1.el7.x86_64.rpm
rpm -if clinfo-2.1.17.02.09-1.el7.x86_64.rpm
rpm -if ocl-icd-2.2.12-1.el7.x86_64.rpm
clinfo

mkdir -p /etc/OpenCL/vendors && 
  echo "libnvidia-opencl.so.1" > /etc/OpenCL/vendors/nvidia.icd
```

Looking at Driverless AI log files

```
less log/dai.log
less log/h2o.log
less log/procsy.log
less log/vis-server.log
```
Stopping Driverless AI

```
# Stop Driverless AI.
./kill-dai.sh
```

Uninstalling Driverless AI

To uninstall Driverless AI, just remove the directory created by the unpacking process. By default, all files for Driverless AI are contained within this directory.

Upgrading Driverless AI

**WARNING**: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

1. Stop your previous version of Driverless AI.
2. Run the self-extracting archive for the new version of Driverless AI.
3. Port any previous changes you made to your config.toml file to the newly unpacked directory.
4. Copy the tmp directory (which contains all the Driverless AI working state) from your previous Driverless AI installation into the newly unpacked directory.
5. Start your newly extracted version of Driverless AI.

5.3.5 Linux in the Cloud

To simplify cloud installation, Driverless AI is provided as an AMI for the following cloud platforms:

- AWS AMI
- Azure Image
- Google Cloud

The installation steps for AWS, Azure, and Google Cloud assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/driverless-ai/](https://www.h2o.ai/driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.
Install on AWS

Driverless AI can be installed on Amazon AWS using one of two methods:

Install the Driverless AI AWS Marketplace AMI

A Driverless AI AMI is available in the AWS Marketplace beginning with Driverless AI version 1.5.2. This section describes how to install and run Driverless AI through the AWS Marketplace.

Environment

<table>
<thead>
<tr>
<th>Provider</th>
<th>Instance Type</th>
<th>Num GPUs</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>p2.xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>p2.8xlarge</td>
<td>8</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p2.16xlarge</td>
<td>16</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p3.2xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>p3.8xlarge</td>
<td>4</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p3.16xlarge</td>
<td>8</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>g3.4xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>g3.8xlarge</td>
<td>2</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>g3.16xlarge</td>
<td>4</td>
<td>Serious use</td>
</tr>
</tbody>
</table>

Installation Procedure

1. Log in to the AWS Marketplace.
2. Search for Driverless AI.
3. Select the version of Driverless AI that you want to install.
4. Scroll down to review/edit your region and the selected infrastructure and pricing.

Pricing Information

Use this tool to estimate the software and infrastructure costs based on your configuration choices. Your usage and costs might be different from this estimate. They will be reflected on your monthly AWS billing reports.

5. Return to the top and select Continue to Subscribe.
6. Review the subscription, then click Continue to Configure.

7. If desired, change the Fulfillment Option, Software Version, and Region. Note that this page also includes the AMI ID for the selected software version. Click Continue to Launch when you are done.

8. Review the configuration and choose a method for launching Driverless AI. Be sure to also review the Usage Instructions. This button provides you with the login and password for launching Driverless AI. Scroll down to the bottom of the page and click Launch when you are done.
Using Driverless AI, Release 1.6.2

Launch this software

Review your configuration and choose how you wish to launch the software.

**Configuration Details**

- **Fulfillment Option**: 64-bit (x86) Amazon Machine Image (AMI)
- **H2O Driverless AI**
- **Software Version**: 1.5.4
- **Region**: US East (N. Virginia)

[Usage instructions]

**Choose Action**

Choose this action to launch from this website

Launch from Website

You will receive a “Success” message when the image launches successfully.
Starting Driverless AI

This section describes how to start Driverless AI after the Marketplace AMI has been successfully launched.

1. Navigate to the EC2 Console.
2. Select your instance.
3. Open another browser and launch Driverless AI by navigating to https://<public IP of the instance>:12345.
4. Sign in to Driverless AI with the username and password provided in the Usage Instructions. You will be prompted to enter your Driverless AI license key the first time that you log in.

Stopping the EC2 Instance

The EC2 instance will continue to run even when you close the aws.amazon.com portal. To stop the instance:

1. On the EC2 Dashboard, click the Running Instances link under the Resources section.
2. Select the instance that you want to stop.
3. In the Actions drop down menu, select Instance State > Stop.
4. A confirmation page will display. Click Yes, Stop to stop the instance.
Upgrading the Driverless AI Marketplace Image

Note that the first offering of the Driverless AI Marketplace image was 1.5.2. As such, it is only possible to upgrade to versions greater than that.

Perform the following steps if you are upgrading to a Driverless AI Marketplace image version greater than 1.5.2. Replace `dai_NEWVERSION.deb` below with the new Driverless AI version (for example, `dai_1.5.4_amd64.deb`). Note that this upgrade process inherits the service user and group from `/etc/dai/User.conf` and `/etc/dai/Group.conf`. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade Driverless AI.
sudo dpkg -i dai_NEWVERSION.deb
sudo systemctl daemon-reload
sudo systemctl start dai
```

Install the Driverless AI AWS Community AMI

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

Environment

<table>
<thead>
<tr>
<th>Provider</th>
<th>Instance Type</th>
<th>Num GPUs</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>p2.xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>p2.8xlarge</td>
<td>8</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p2.16xlarge</td>
<td>16</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p3.2xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>p3.8xlarge</td>
<td>4</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>p3.16xlarge</td>
<td>8</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>g3.4xlarge</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>g3.8xlarge</td>
<td>2</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>g3.16xlarge</td>
<td>4</td>
<td>Serious use</td>
</tr>
</tbody>
</table>

Installing the EC2 Instance

2. In the upper right corner of the Amazon Web Services page, make sure that the location drop-down is US East (N Virginia).
3. Select the EC2 option under the Compute section to open the EC2 Dashboard.
4. Click the **Launch Instance** button under the Create Instance section.

5. Under Community AMIs, search for **h2oai**, and then select the version that you want to launch.
Using Driverless AI, Release 1.6.2

6. On the Choose an Instance Type page, select GPU compute in the Filter by dropdown. This will ensure that your Driverless AI instance will run on GPUs. Select a GPU compute instance from the available options. (We recommend at least 32 vCPUs.) Click the Next: Configure Instance Details button.

7. Specify the Instance Details that you want to configure. Create a VPC or use an existing one, and ensure that “Auto-Assign Public IP” is enabled and associated to your subnet. Click Next: Add Storage.
8. Specify the Storage Device settings. Note again that Driverless AI requires 10 GB to run and will stop working if less than 10 GB is available. The machine should have a minimum of 30 GB of disk space. Click Next: Add Tags.

9. If desired, add unique Tag name to identify your instance. Click Next: Configure Security Group.

10. Add the following security rules to enable SSH access to Driverless AI, then click Review and Launch.

<table>
<thead>
<tr>
<th>Type</th>
<th>Protocol</th>
<th>Port Range</th>
<th>Source</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSH</td>
<td>TCP</td>
<td>22</td>
<td>Anywhere 0.0.0.0/0</td>
<td></td>
</tr>
<tr>
<td>Custom TCP Rule</td>
<td>TCP</td>
<td>12345</td>
<td>Anywhere 0.0.0.0/0</td>
<td>Launch DAI</td>
</tr>
</tbody>
</table>
11. Review the configuration, and then click **Launch**.

12. A popup will appear prompting you to select a key pair. This is required in order to SSH into the instance. You can select your existing key pair or create a new one. Be sure to accept the acknowledgement, then click **Launch Instances** to start the new instance.

13. Upon successful completion, a message will display informing you that your instance is launching. Click the **View Instances** button to see information about the instance including the IP address. The **Connect** button on this page provides information on how to SSH into your instance.

14. Open a Terminal window and SSH into the IP address of the AWS instance. Replace the DNS name below with your instance DNS.

   ```
   ssh -i "mykeypair.pem" ubuntu@ec2-34-230-6-230.compute-1.amazonaws.com
   ```

15. If you selected a GPU-compute instance, then you must enable persistence and optimizations of the GPU. The
Using Driverless AI, Release 1.6.2

Commands vary depending on the instance type. Note also that these commands need to be run once every reboot. Refer to the following for more information:

- https://www.migenius.com/articles/realityserver-on-aws

```
# g3:
sudo nvidia-smi -pm 1
sudo nvidia-smi -acp 0
sudo nvidia-smi --auto-boost-permission=0
sudo nvidia-smi --auto-boost-default=0
sudo nvidia-smi -ac "2505,1177"

# p2:
sudo nvidia-smi -pm 1
sudo nvidia-smi -acp 0
sudo nvidia-smi --auto-boost-permission=0
sudo nvidia-smi --auto-boost-default=0
sudo nvidia-smi -ac "2505,875"

# p3:
sudo nvidia-smi -pm 1
sudo nvidia-smi -acp 0
sudo nvidia-smi -ac "877,1530"
```

16. At this point, you can copy data into the data directory on the host machine using `scp`. (Note that the data folder already exists.) For example:

```
scp <data_file>.csv ubuntu@ec2-34-230-6-230.compute-1.amazonaws.com:/home/→data
```

The data will be visible inside the Docker container.

17. Connect to Driverless AI with your browser. You will be prompted to enter your Driverless AI license key the first time that you log in.

```
```

**Stopping the EC2 Instance**

The EC2 instance will continue to run even when you close the aws.amazon.com portal. To stop the instance:

1. On the EC2 Dashboard, click the **Running Instances** link under the Resources section.
2. Select the instance that you want to stop.
3. In the **Actions** drop down menu, select **Instance State > Stop**.
4. A confirmation page will display. Click **Yes, Stop** to stop the instance.

**Upgrading the Driverless AI Community Image**

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.
Using Driverless AI, Release 1.6.2

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

Upgrading from Version 1.2.2 or Earlier

The following example shows how to upgrade from 1.2.2 or earlier to the current version. Upgrading from these earlier versions requires an edit to the `start` and `h2oai` scripts.

1. SSH into the IP address of the image instance and copy the existing experiments to a backup location:

   ```
   # Set up a directory of the previous version name
   mkdir dai_rel_1.2.2
   # Copy the data, log, license, and tmp directories as backup
   cp -r ./data dai_rel_1.2.2/data
   cp -r ./log dai_rel_1.2.2/log
   cp -r ./license dai_rel_1.2.2/license
   cp -r ./tmp dai_rel_1.2.2/tmp
   ```

2. `wget` the newer image. The command below retrieves version 1.2.2:

   ```
   wget https://s3.amazonaws.com/artifacts.h2o.ai/releases/ai/h2o/dai/rel-1.2.2-6/x86_64-centos7/dai-docker-centos7-x86_64-1.2.2-9.0.tar.gz
   ```

3. In the `/home/ubuntu/scripts/` folder, edit both the `start.sh` and `h2oai.sh` scripts to use the newer image.

4. Use the `docker load` command to load the image:

   ```
   docker load < ami-0c50db5e1999408a7
   ```

5. Optionally run `docker images` to ensure that the new image is in the registry.


Upgrading from Version 1.3.0 or Later

The following example shows how to upgrade from version 1.3.0.

1. SSH into the IP address of the image instance and copy the existing experiments to a backup location:

   ```
   # Set up a directory of the previous version name
   mkdir dai_rel_1.3.0
   # Copy the data, log, license, and tmp directories as backup
   cp -r ./data dai_rel_1.3.0/data
   ```

(continues on next page)
2. `wget` the newer image. Replace `VERSION` and `BUILD` below with the Driverless AI version.

```
wget https://s3.amazonaws.com/artifacts.h2o.ai/releases/ai/h2o/dai/VERSION-BUILD/x86_64-centos7/dai-docker-centos7-x86_64-VERSION.tar.gz
```

3. Use the `docker load` command to load the image:

```
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

4. In the new AMI, locate the `DAI_RELEASE` file, and edit that file to match the new image tag.

5. Stop and then start Driverless AI.

```
h2oai stop
h2oai start
```

When installing via AWS, you can also enable role-based authentication.

### AWS Role-Based Authentication

In Driverless AI, it is possible to enable role-based authentication via the IAM role. This is a two-step process that involves setting up AWS IAM and then starting Driverless AI by specifying the role in the config.toml file or by setting the `AWS_USE_EC2_ROLE_CREDENTIALS` environment variable to `True`.

#### AWS IAM Setup

1. Create an IAM role. This IAM role should have a Trust Relationship with Principal Trust Entity set to your Account ID. For example: trust relationship for Account ID 524466471676 would look like:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": {
            "AWS": "arn:aws:iam::524466471676:root"
         },
         "Action": "sts:AssumeRole"
      }
   ]
}
```
2. Create a new policy that allows users to assume the role:

3. Assign the policy to the user.

Driverless AI Setup

Update the `aws_use_ec2_role_credentials` config variable in the config.toml file or start Driverless AI using the `AWS_USE_EC2_ROLE_CREDENTIALS` environment variable.

Resources

2. Creating a Role to Delegate Permissions to an IAM User: https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-user.html
3. Assuming an IAM Role in the AWS CLI: https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-role.html

Install on Azure

This section describes how to install the Driverless AI image from Azure.

Note: Prior versions of the Driverless AI installation and upgrade on Azure were done via Docker. This is no longer the case as of version 1.5.2.

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.
Environment

<table>
<thead>
<tr>
<th>Provider</th>
<th>Instance Type</th>
<th>Num GPUs</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Azure</td>
<td>Standard_NV6</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>Standard_NV12</td>
<td>2</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>Standard_NV24</td>
<td>4</td>
<td>Serious use</td>
</tr>
<tr>
<td></td>
<td>Standard_NC6</td>
<td>1</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>Standard_NC12</td>
<td>2</td>
<td>Experimentation</td>
</tr>
<tr>
<td></td>
<td>Standard_NC24</td>
<td>4</td>
<td>Serious use</td>
</tr>
</tbody>
</table>

About the Install

• The ‘dai’ service user is created locally (in /etc/passwd) if it is not found by ‘getent passwd’. You can override
  the user by providing the DAI_USER environment variable during rpm or dpkg installation.

• The ‘dai’ service group is created locally (in /etc/group) if it is not found by ‘getent group’. You can override
  the group by providing the DAI_GROUP environment variable during rpm or dpkg installation.

• Configuration files are put in /etc/dai and owned by the ‘root’ user:
  – /etc/dai/config.toml: Driverless AI config file (See Using the config.toml File section for details)
  – /etc/dai/User.conf: Systemd config file specifying the service user
  – /etc/dai/Group.conf: Systemd config file specifying the service group
  – /etc/dai/EnvironmentFile.conf: Systemd config file specifying (optional) environment variable overrides

• Software files are put in /opt/h2oai/dai and owned by the ‘root’ user

• The following directories are owned by the service user so they can be updated by the running software:
  – /opt/h2oai/dai/home: The application’s home directory (license key files are stored here)
  – /opt/h2oai/dai/tmp: Experiments and imported data are stored here
  – /opt/h2oai/dai/log: Log files go here if you are not using systemd (if you are using systemd, then the use
    the standard journalctl tool)

• By default, Driverless AI looks for a license key in /opt/h2oai/dai/home/.driverlessai/license.sig. If you are
  installing Driverless AI programmatically, you can copy a license key file to that location. If no license key is
  found, the application will interactively guide you to add one from the Web UI.

• systemd unit files are put in /usr/lib/systemd/system

• Symbolic links to the configuration files in /etc/dai files are put in /etc/systemd/system

If your environment is running an operational systemd, that is the preferred way to manage Driverless AI. The package
installs the following systemd services and a wrapper service:

• dai: Wrapper service that starts/stops the other three services

• dai-dai: Main Driverless AI process

• dai-h2o: H2O-3 helper process used by Driverless AI

• dai-procsy: Procsy helper process used by Driverless AI

• dai-vis-server: Visualization server helper process used by Driverless AI

If you don’t have systemd, you can also use the provided run script to start Driverless AI.
Installing the Azure Instance

1. Log in to your Azure portal at https://portal.azure.com, and click the Create a Resource button.

2. Search for and select H2O DriverlessAI in the Marketplace.

3. Click Create. This launches the H2O DriverlessAI Virtual Machine creation process.

4. On the Basics tab:
   a. Enter a name for the VM.
   b. Select the Disk Type for the VM. Use HDD for GPU instances.
   c. Enter the name that you will use when connecting to the machine through SSH.
d. Enter and confirm a password that will be used when connecting to the machine through SSH.

e. Specify the Subscription option. (This should be Pay-As-You-Go.)

f. Enter a name unique name for the resource group.

g. Specify the VM region.

Click OK when you are done.

5. On the Size tab, select your virtual machine size. Specify the HDD disk type and select a configuration. We recommend using an N-Series type, which comes with a GPU. Also note that Driverless AI requires 10 GB of free space in order to run and will stop working of less than 10 GB is available. We recommend a minimum of 30 GB of disk space. Click OK when you are done.
6. On the **Settings** tab, select or create the Virtual Network and Subnet where the VM is going to be located and then click **OK**.
7. The **Summary** tab performs a validation on the specified settings and will report back any errors. When the validation passes successfully, click **Create** to create the VM.

8. After the VM is created, it will be available under the list of Virtual Machines. Select this Driverless AI VM to view the IP address of your newly created machine.

9. Connect to Driverless AI with your browser using the IP address retrieved in the previous step.

   ```
   ```

**Stopping the Azure Instance**

The Azure instance will continue to run even when you close the Azure portal. To stop the instance:

1. Click the **Virtual Machines** left menu item.
2. Select the checkbox beside your DriverlessAI virtual machine.
3. On the right side of the row, click the **…** button, then select **Stop**. (Note that you can then restart this by selecting **Start**.)
Upgrading the Driverless AI Image

**WARNING**: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

Upgrading from Version 1.2.2 or Earlier

It is not possible to upgrade from version 1.2.2 or earlier to the latest version. You have to manually remove the 1.2.2 container and then reinstall the latest Driverless AI version. Be sure to back up your data before doing this.

Upgrading from Version 1.3.0 to 1.5.1

1. SSH into the IP address of the image instance and copy the existing experiments to a backup location:

   ```bash
   # Set up a directory of the previous version name
   mkdir dai_rel_1.3.0
   ```

(continues on next page)
# Copy the data, log, license, and tmp directories as backup

cp -r ./data dai_rel_1.3.0/data

cp -r ./log dai_rel_1.3.0/log

cp -r ./license dai_rel_1.3.0/license

cp -r ./tmp dai_rel_1.3.0/tmp

2. `wget` the newer image. Replace VERSION and BUILD below with the Driverless AI version.

```
wget https://s3.amazonaws.com/artifacts.h2o.ai/releases/ai/h2o/VERSION-BUILD/x86_64-centos7/dai-docker-centos7-x86_64-VERSION.tar.gz
```

3. Use the `docker load` command to load the image:

```
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

4. Run `docker images` to find the new image tag.

5. Start the Driverless AI Docker image with `nvidia-docker` and replace TAG below with the image tag:

```
# Start the Driverless AI Docker image
nvidia-docker run \\
    --pid=host \\
    --init \ 
    --rm \\
    --shm-size=256m \\
    -u `id -u`:`id -g` \\
    -p 12345:12345 \\
    -v `pwd`/data:/data \\
    -v `pwd`/log:/log \\
    -v `pwd`/license:/license \\
    -v `pwd`/tmp:/tmp \\
    h2oai/dai-centos7-x86_64:TAG
```

---

**Upgrading from version 1.5.2 or Later**

Upgrading to versions 1.5.2 and later is no longer done via Docker. Instead, perform the following steps if you are upgrading to version 1.5.2 or later. Replace `dai_NEWVERSION.deb` below with the new Driverless AI version (for example, `dai_1.6.1_amd64.deb`). Note that this upgrade process inherits the service user and group from `/etc/dai/User.conf` and `/etc/dai/Group.conf`. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

```
# Stop Driverless AI.
sudo systemctl stop dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade Driverless AI.
sudo dpkg -i dai_NEWVERSION.deb
sudo systemctl daemon-reload
sudo systemctl start dai
```

**Install on Google Compute**

Driverless AI can be installed on Google Compute using one of two methods:
- Install the Google Cloud Platform offering. This installs Driverless AI via the available Cloud Launcher offering.
- Install and Run in a Docker Container on Google Compute Engine. This installs and runs Driverless AI from scratch in a Docker container on Google Compute Engine.

Select your desired installation procedure below:

**Install the Google Cloud Platform Offering**

This section describes how to install and start Driverless AI in a Google Compute environment using the available Cloud Launcher offering. This assumes that you already have a Google Cloud Platform account. If you don’t have an account, go to https://console.cloud.google.com/getting-started to create one.

**Before You Begin**

If you are trying GCP for the first time and have just created an account, please check your Google Compute Engine (GCE) resource quota limits. By default, GCP allocates a maximum of 8 CPUs and no GPUs. Our default recommendation for launching Driverless AI is 32 CPUs, 120 GB RAM, and 2 P100 NVIDIA GPUs. You can change these settings to match your quota limit, or you can request more resources from GCP. Refer to https://cloud.google.com/compute/quotas for more information, including information on how to check your quota and request additional quota.

**Installation Procedure**

1. In your browser, log in to the Google Compute Engine Console at https://console.cloud.google.com/.
2. In the left navigation panel, select Cloud Launcher.

3. On the Cloud Launcher page, search for **Driverless** and select the H2O.ai Driverless AI offering. The following page will display.
4. Click **Launch on Compute Engine**. (If necessary, refer to Google Compute Instance Types for information about machine and GPU types.)

   - Select a zone that has p100s or k80s (such as us-east1-)
   - Optionally change the number of cores and amount of memory. (This defaults to 32 CPUs and 120 GB RAM.)
   - Specify a GPU type. (This defaults to a p100 GPU.)
   - Optionally change the number of GPUs. (Default is 2.)
   - Specify the boot disk type and size.
   - Optionally change the network name and subnetwork names. Be sure that whichever network you specify has port 12345 exposed.
   - Click **Deploy** when you are done. Driverless AI will begin deploying. Note that this can take several minutes.
5. A summary page displays when the compute engine is successfully deployed. This page includes the instance ID and the username (always h2oai) and password that will be required when starting Driverless AI. Click on the Instance link to retrieve the external IP address for starting Driverless AI.

5.3. Linux X86_64 Installs
6. In your browser, go to https://[External_IP]:12345 to start Driverless AI.

7. Agree to the Terms and Conditions.

8. Log in to Driverless AI using your user name and password.

9. Optionally enable GCS and Big Query access.
   
a. In order to enable GCS and Google BigQuery access, you must pass the running instance a service account json file configured with GCS and GBQ access. The Driverless AI image comes with a blank file called `service_account.json`. Obtain a functioning service account json file from GCP, rename it to “service_account.json”, and copy it to the Ubuntu user on the running instance.

```
gcloud compute scp /path/to/service_account.json ubuntu@<running_instance_name>:service_account.json
```

b. SSH into the machine running Driverless AI, and verify that the service_account.json file is in the `/etc/dai/` folder.

c. Restart the machine for the changes to take effect.

```
sudo systemctl stop dai
# Wait for the system to stop

# Verify that the system is no longer running
sudo systemctl status dai

# Restart the system
sudo systemctl start dai
```
Upgrading the Google Cloud Platform Offering

Perform the following steps to upgrade the Driverless AI Google Platform offering. Replace `dai_NEWVERSION.deb` below with the new Driverless AI version (for example, `dai_1.6.1_amd64.deb`). Note that this upgrade process inherits the service user and group from `/etc/dai/User.conf` and `/etc/dai/Group.conf`. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

```
# Stop Driverless AI.
sudo systemctl stop dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade Driverless AI.
sudo dpkg -i dai_NEWVERSION.deb
sudo systemctl daemon-reload
sudo systemctl start dai
```

Install and Run in a Docker Container on Google Compute Engine

This section describes how to install and start Driverless AI from scratch using a Docker container in a Google Compute environment.

This installation assumes that you already have a Google Cloud Platform account. If you don’t have an account, go to https://console.cloud.google.com/getting-started to create one. In addition, refer to Google’s Machine Types documentation for information on Google Compute machine types.

Watch the installation video here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

Before You Begin

If you are trying GCP for the first time and have just created an account, please check your Google Compute Engine (GCE) resource quota limits. By default, GCP allocates a maximum of 8 CPUs and no GPUs. You can change these settings to match your quota limit, or you can request more resources from GCP. Refer to https://cloud.google.com/compute/quotas for more information, including information on how to check your quota and request additional quota.

Installation Procedure

1. In your browser, log in to the Google Compute Engine Console at https://console.cloud.google.com/.
2. In the left navigation panel, select Compute Engine > VM Instances.
3. Click **Create Instance**.

4. Specify the following at a minimum:
   - A unique name for this instance.
   - The desired **zone**. Note that not all zones and user accounts can select zones with GPU instances. Refer to the following for information on how to add GPUs: [https://cloud.google.com/compute/docs/gpus/](https://cloud.google.com/compute/docs/gpus/).
   - A supported OS, for example Ubuntu 16.04. Be sure to also increase the disk size of the OS image to be 64 GB.

   Click **Create** at the bottom of the form when you are done. This creates the new VM instance.
5. Create a Firewall rule for Driverless AI. On the Google Cloud Platform left navigation panel, select VPC network > Firewall rules. Specify the following settings:

- Specify a unique name and Description for this instance.
- Change the Targets dropdown to All instances in the network.
- Specify the Source IP ranges to be 0.0.0.0/0.
- Under Protocols and Ports, select Specified protocols and ports and enter the following: tcp:12345.

Click Create at the bottom of the form when you are done.
6. On the VM Instances page, SSH to the new VM Instance by selecting **Open in Browser Window** from the SSH dropdown.

7. H2O provides a script for you to run in your VM instance. Open an editor in the VM instance (for example, `vi`). Copy one of the scripts below (depending on whether you are running GPUs or CPUs). Save the script as `install.sh`.

```
# SCRIPT FOR GPUs ONLY
apt-get -y update
apt-get -y --no-install-recommends install \
curl \
apt-utils \
(continues on next page)
```
Using Driverless AI, Release 1.6.2

(continued from previous page)

```bash
python-software-properties \
software-properties-common

add-apt-repository -y ppa:graphics-drivers/ppa
add-apt-repository -y "deb [arch=amd64] https://download.docker.com/linux/\n  →ubuntu $(lsb_release -cs) stable"
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -

apt-get update
apt-get install -y 
  nvidia-384 
  nvidia-modprobe 
  docker-ce
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | \\n  sudo apt-key add
distribution=$(./etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-\n  →docker.list | \\
  sudo tee /etc/apt/sources.list.d/nvidia-docker.list
sudo apt-get update
# Install nvidia-docker2 and reload the Docker daemon configuration
sudo apt-get install -y nvidia-docker2

# SCRIPT FOR CPUs ONLY
apt-get -y update
apt-get -y --no-install-recommends install \
  curl \n  apt-utils \n  python-software-properties \n  software-properties-common

add-apt-repository -y "deb [arch=amd64] https://download.docker.com/linux/\n  →ubuntu $(lsb_release -cs) stable"
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | apt-key add -
apt-get update
apt-get install -y docker-ce
```

8. Type the following commands to run the install script.

```bash
chmod +x install.sh
sudo ./install.sh
```

9. In your user folder, create the following directories as your user.

```bash
mkdir ~/tmp
mkdir ~/log
mkdir ~/data
mkdir ~/scripts
mkdir ~/license
mkdir ~/demo
mkdir -p ~/jupyter/notebooks
```

10. Add your Google Compute user name to the Docker container.
Using Driverless AI, Release 1.6.2

```bash
sudo usermod -aG docker <username>
```

11. Reboot the system to enable NVIDIA drivers.
```bash
sudo reboot
```

12. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.
```bash
sudo docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

13. Load the Driverless AI Docker image. The following example shows how to load Driverless AI. Replace VERSION with your image.
```bash
sudo docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

14. If you are running CPUs, you can skip this step. Otherwise, you must enable persistence of the GPU. Note that this needs to be run once every reboot. Refer to the following for more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.
```bash
sudo nvidia-persistenced --user <USER>
sudo nvidia-smi -pm 1
```

15. Start the Driverless AI Docker image with `nvidia-docker run` (GPUs) or `docker run` (CPUs). Note that you must have write privileges for the folders that are created below. You can replace ‘pwd’ with the path to /home/<username> or start with `sudo nvidia-docker run`. Replace TAG with the Docker image tag (run `docker images` if necessary.) Also, refer to Using Data Connectors with the Docker Image for information on how to add the GCS and GBQ data connectors to your Driverless AI instance.
```bash
# Start the Driverless AI Docker image
nvidia-docker run \
  --pid=host \
  --init \
  --rm \
  --shm-size=256m \
  -u `id -u`:`id -g` \
  -p 12345:12345 \
  -v `pwd`/data:/data \
  -v `pwd`/log:/log \
  -v `pwd`/license:/license \
  -v `pwd`/tmp:/tmp \
  h2oai/dai-centos7-x86_64:TAG
```

Driverless AI will begin running:
```
--------------
Welcome to H2O.ai's Driverless AI
--------------
- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```

16. Connect to Driverless AI with your browser:
```bash
```
**Stopping the GCE Instance**

The Google Compute Engine instance will continue to run even when you close the portal. You can stop the instance using one of the following methods:

**Stopping in the browser**

1. On the VM Instances page, click on the VM instance that you want to stop.
2. Click **Stop** at the top of the page.
3. A confirmation page will display. Click **Stop** to stop the instance.

**Stopping in Terminal**

SSH into the machine that is running Driverless AI, and then run the following:

```
h2oai stop
```

**Upgrading Driverless AI**

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

**Note:** Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:

   ```
   # Set up directory with the version name
   mkdir dai_rel_VERSION
   # cd into the new directory
   cd dai_rel_VERSION
   ```

3. Retrieve the Driverless AI package from [https://www.h2o.ai/download/](https://www.h2o.ai/download/) and add it to the new directory.
4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace VERSION with your image.

   ```
   # Load the Driverless AI docker image
   docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
   ```
5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:

```
# Copy the data, log, license, and tmp directories on the host machine
cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
```

At this point, your experiments from the previous versions will be visible inside the Docker container.

6. Use `docker images` to find the new image tag.

7. Start the Driverless AI Docker image.


### 5.4 IBM Power Installs

This section provides installation steps for IBM Power environments. This includes information for Docker image installs, RPMs, Deb, and Tar installs.

**Note:** OpenCL and LightGBM with GPUs are not supported on Power currently

#### 5.4.1 IBM Docker Images

To simplify local installation, Driverless AI is provided as a Docker image for the following system combination:

<table>
<thead>
<tr>
<th>Host OS</th>
<th>Docker Version</th>
<th>Host Architecture</th>
<th>Min Mem</th>
</tr>
</thead>
<tbody>
<tr>
<td>RHEL or CentOS 7.4 or later</td>
<td>Docker CE</td>
<td>ppc64le</td>
<td>64 GB</td>
</tr>
</tbody>
</table>

**Notes:**

- Cuda 9 or later with NVIDIA drivers >= 396
- OpenCL and LightGBM with GPUs are not supported on Power currently.

For the best performance, including GPU support, use nvidia-docker2. For a lower-performance experience without GPUs, use regular docker (with the same docker image).

These installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/products/h2o-driverless-ai/](https://www.h2o.ai/products/h2o-driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

### Install on IBM with GPUs

This section describes how to install and start the Driverless AI Docker image on RHEL for IBM Power LE systems with GPUs. Note that nvidia-docker has limited support for ppc64le machines. More information about nvidia-docker support for ppc64le machines is available here.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Retrieve the Driverless AI Docker image from [https://www.h2o.ai/download/](https://www.h2o.ai/download/).
2. Install and start Docker CE.

```bash
sudo yum install -y yum-utils
sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/
  docker-ce.repo
sudo yum makecache fast
sudo yum -y install docker-ce
sudo systemctl start docker
```


```bash
distribution=$(./etc/os-release;echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/nvidia-
  docker.repo | \
  sudo tee /etc/yum.repos.d/nvidia-docker.repo
sudo yum install nvidia-docker2
```

4. Verify that the NVIDIA driver is up and running. If the driver is not up and running, log on to http://www.nvidia.com/Download/index.aspx?lang=en-us to get the latest NVIDIA Tesla V/P/K series driver.

```bash
nvidia-docker run --rm nvidia/cuda nvidia-smi
```

5. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

```bash
# Set up directory with the version name
mkdir dai_rel_VERSION
```

6. Change directories to the new folder, then load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI. Replace VERSION with your image.

```bash
# cd into the new directory
cd dai_rel_VERSION

# Load the Driverless AI docker image
docker load < dai-docker-centos7-ppc64le-VERSION.tar.gz
```

7. Enable persistence of the GPU. Note that this needs to be run once every reboot. Refer to the following for more information: http://docs.nvidia.com/deploy/driver-persistence/index.html.

```bash
nvidia-persistenced --user <USER>
nvidia-smi -pm 1
```

8. Set up the data, log, and license directories on the host machine (within the new directory):

```bash
# Set up the data, log, license, and tmp directories on the host machine
mkdir data
mkdir log
mkdir license
mkdir tmp
```

9. At this point, you can copy data into the data directory on the host machine. The data will be visible inside the Docker container.

10. Run `docker images` to find the image tag.

11. Start the Driverless AI Docker image with nvidia-docker and replace TAG below with the image tag:

```bash
```
# Start the Driverless AI Docker image
nvidia-docker run \
    --pid=host \n    --init \n    --rm \n    --shm-size=256m \n    -u `id -u`:`id -g` \n    -p 12345:12345 \n    -v `pwd`/data:/data \n    -v `pwd`/log:/log \n    -v `pwd`/license:/license \n    -v `pwd`/tmp:/tmp \n    h2oai/dai-centos7-ppc64le:TAG

Driverless AI will begin running:

```
Welcome to H2O.ai's Driverless AI
```

- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container


## Install on IBM with CPUs

This section describes how to install and start the Driverless AI Docker image on RHEL for IBM Power LE systems with CPUs. Note that this uses Docker EE and not NVIDIA Docker. GPU support will not be available.

### Watch the installation video [here](#). Note that some of the images in this video may change between releases, but the installation steps remain the same.

### Note: As of this writing, Driverless AI has only been tested on RHEL version 7.4.

Open a Terminal and ssh to the machine that will run Driverless AI. Once you are logged in, perform the following steps.

1. Install and start Docker CE.

   ```sh
   sudo yum install -y yum-utils
   sudo yum-config-manager --add-repo https://download.docker.com/linux/centos/    
   →docker-ce.repo
   sudo yum makecache fast
   sudo yum -y install docker-ce
   sudo systemctl start docker
   ```

2. On the machine that is running Docker EE, retrieve the Driverless AI Docker image from [https://www.h2o.ai/driverless-ai-download/](#).

3. Set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

   ```sh
   # Set up directory with the version name
   mkdir dai_rel_VERSION
   ```
4. Load the Driverless AI Docker image inside the new directory. The following example shows how to load Driverless AI. Replace VERSION with your image.

```
# Load the Driverless AI Docker image
docker load < dai-docker-centos7-ppc64le-VERSION.tar.gz
```

5. Set up the data, log, license, and tmp directories (within the new directory):

```
# cd into the directory associated with the selected version of Driverless AI
cd dai_rel_VERSION

# Set up the data, log, license, and tmp directories on the host machine
mkdir data
mkdir log
mkdir license
mkdir tmp
```

6. Copy data into the `data` directory on the host. The data will be visible inside the Docker container at `/<user-home>/data`.

7. Run `docker images` to find the image tag.

8. Start the Driverless AI Docker image and replace TAG below with the image tag. Note that GPU support will not be available.

```
$ docker run 
   --pid=host 
   --init 
   --rm 
   -u `id -u:;id -g` 
   -p 12345:12345 
   -v `pwd`/data:/data 
   -v `pwd`/log:/log 
   -v `pwd`/license:/license 
   -v `pwd`/tmp:/tmp 
   h2oai/dai-centos7-ppc64le:TAG
```

Driverless AI will begin running:
```
-------------------------------
Welcome to H2O.ai's Driverless AI
-------------------------------
- Put data in the volume mounted at /data
- Logs are written to the volume mounted at /log/20180606-044258
- Connect to Driverless AI on port 12345 inside the container
- Connect to Jupyter notebook on port 8888 inside the container
```


**Stopping the Docker Image**

To stop the Driverless AI Docker image, type `Ctrl + C` in the Terminal (Mac OS X) or PowerShell (Windows 10) window that is running the Driverless AI Docker image.
Upgrading the Docker Image

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

**WARNING**: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

**Note**: Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:

   ```
   # Set up directory with the version name
   mkdir dai_rel_VERSION
   # cd into the new directory
   cd dai_rel_VERSION
   ```

3. Retrieve the Driverless AI package from https://www.h2o.ai/download/ and add it to the new directory.
4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace VERSION with your image.

   ```
   # Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
   ```

5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:

   ```
   # Copy the data, log, license, and tmp directories on the host machine
cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
   ```

   At this point, your experiments from the previous versions will be visible inside the Docker container.

6. Use `docker images` to find the new image tag.
7. Start the Driverless AI Docker image.
5.4.2 IBM RPMs

For IBM machines that will not use the Docker image or DEB, an RPM installation is available for ppc64le RHEL 7. The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit https://www.h2o.ai/products/h2o-driverless-ai/. Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

Note: OpenCL and LightGBM with GPUs are not supported on Power currently.

Requirements

- RedHat 7
- Cuda 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- Driverless AI RPM, available from https://www.h2o.ai/download/

About the Install

- The ‘dai’ service user is created locally (in /etc/passwd) if it is not found by ‘getent passwd’. You can override the user by providing the DAI_USER environment variable during rpm or dpkg installation.
- The ‘dai’ service group is created locally (in /etc/group) if it is not found by ‘getent group’. You can override the group by providing the DAI_GROUP environment variable during rpm or dpkg installation.
- Configuration files are put in /etc/dai and owned by the ‘root’ user:
  - /etc/dai/config.toml: Driverless AI config file (See Using the config.toml File section for details)
  - /etc/dai/User.conf: Systemd config file specifying the service user
  - /etc/dai/Group.conf: Systemd config file specifying the service group
  - /etc/dai/EnvironmentFile.conf: Systemd config file specifying (optional) environment variable overrides
- Software files are put in /opt/h2oai/dai and owned by the ‘root’ user
- The following directories are owned by the service user so they can be updated by the running software:
  - /opt/h2oai/dai/home: The application’s home directory (license key files are stored here)
  - /opt/h2oai/dai/tmp: Experiments and imported data are stored here
  - /opt/h2oai/dai/log: Log files go here if you are not using systemd (if you are using systemd, then the use the standard journalctl tool)
- By default, Driverless AI looks for a license key in /opt/h2oai/dai/home/.driverlessai/license.sig. If you are installing Driverless AI programmatically, you can copy a license key file to that location. If no license key is found, the application will interactively guide you to add one from the Web UI.
- systemd unit files are put in /usr/lib/systemd/system
- Symbolic links to the configuration files in /etc/dai files are put in /etc/systemd/system

If your environment is running an operational systemd, that is the preferred way to manage Driverless AI. The package installs the following systemd services and a wrapper service:

- **dai**: Wrapper service that starts/stops the other three services
Using Driverless AI, Release 1.6.2

- **dai-dai**: Main Driverless AI process
- **dai-h2o**: H2O-3 helper process used by Driverless AI
- **dai-procsy**: Procsy helper process used by Driverless AI
- **dai-vis-server**: Visualization server helper process used by Driverless AI

If you don’t have systemd, you can also use the provided run script to start Driverless AI.

### Installing Driverless AI

Run the following commands to install the Driverless AI RPM. Replace VERSION with your specific version.

```
# Install Driverless AI.
sudo rpm -i dai-VERSION.rpm
```

By default, the Driverless AI processes are owned by the ‘dai’ user and ‘dai’ group. You can optionally specify a different service user and group as shown below. Replace `<myuser>` and `<mygroup>` as appropriate.

```
# Temporarily specify service user and group when installing Driverless AI.
# rpm saves these for systemd in the /etc/dai/User.conf and /etc/dai/Group.conf files.
sudo DAI_USER=myuser DAI_GROUP=mygroup rpm -i dai-VERSION.rpm
```

You may now optionally make changes to `/etc/dai/config.toml`.

### Starting Driverless AI

If you have systemd (preferred):

```
# Start Driverless AI.
sudo systemctl start dai
```

If you do not have systemd:

```
# Start Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

### Starting NVIDIA Persistence Mode

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: [http://docs.nvidia.com/deploy/driver-persistence/index.html](http://docs.nvidia.com/deploy/driver-persistence/index.html).

```
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```

### Looking at Driverless AI log files

If you have systemd (preferred):

```
sudo systemctl status dai-dai
sudo systemctl status dai-h2o
sudo systemctl status dai-procsy
sudo systemctl status dai-vis-server
```

(continues on next page)
Using Driverless AI, Release 1.6.2

Using Driverless AI, Release 1.6.2

Using Driverless AI, Release 1.6.2

sudo journalctl -u dai-dai
sudo journalctl -u dai-h2o
sudo journalctl -u dai-procsy
sudo journalctl -u dai-vis-server

If you do not have systemd:

sudo less /opt/h2oai/dai/log/dai.log
sudo less /opt/h2oai/dai/log/h2o.log
sudo less /opt/h2oai/dai/log/procsy.log
sudo less /opt/h2oai/dai/log/vis-server.log

Stopping Driverless AI

If you have systemd (preferred):

# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

If you do not have systemd:

# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

Upgrading Driverless AI

WARNING: Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

The upgrade process inherits the service user and group from /etc/dai/User.conf and /etc/dai/Group.conf. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

If you have systemd (preferred):
Using Driverless AI, Release 1.6.2

# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade and restart.
sudo rpm -U dai-NEWVERSION.rpm
sudo systemctl daemon-reload
sudo systemctl start dai

If you do not have systemd:

# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time. If you do not, all previous data will be lost.

# Upgrade and restart.
sudo rpm -U dai-NEWVERSION.rpm
sudo -H -u dai /opt/h2oai/dai/run-dai.sh

Uninstalling Driverless AI

If you have systemd (preferred):

# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall.
sudo rpm -e dai

If you do not have systemd:

# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall.
sudo rpm -e dai

CAUTION! At this point you can optionally completely remove all remaining files, including the database (this cannot be undone):
**5.4.3 IBM DEB**

For IBM machines that will not use the Docker image or RPM, a DEB installation is available for ppc64le Ubuntu 16.04.

**Note**: OpenCL and LightGBM with GPUs are not supported on Power currently.

**Requirements**

- Ubuntu 16.04
- Cuda 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- Driverless AI DEB, available from [https://www.h2o.ai/download/](https://www.h2o.ai/download/)

**About the Install**

- The ‘dai’ service user is created locally (in /etc/passwd) if it is not found by ‘getent passwd’. You can override the user by providing the DAI_USER environment variable during rpm or dpkg installation.
- The ‘dai’ service group is created locally (in /etc/group) if it is not found by ‘getent group’. You can override the group by providing the DAI_GROUP environment variable during rpm or dpkg installation.
- Configuration files are put in `/etc/dai` and owned by the ‘root’ user:
  - `/etc/dai/config.toml`: Driverless AI config file (See *Using the config.toml File* section for details)
  - `/etc/dai/User.conf`: Systemd config file specifying the service user
  - `/etc/dai/Group.conf`: Systemd config file specifying the service group
  - `/etc/dai/EnvironmentFile.conf`: Systemd config file specifying (optional) environment variable overrides
- Software files are put in `/opt/h2oai/dai` and owned by the ‘root’ user
- The following directories are owned by the service user so they can be updated by the running software:
  - `/opt/h2oai/dai/home`: The application’s home directory (license key files are stored here)
  - `/opt/h2oai/dai/tmp`: Experiments and imported data are stored here
  - `/opt/h2oai/dai/log`: Log files go here if you are not using systemd (if you are using systemd, then the use the standard journalctl tool)
- By default, Driverless AI looks for a license key in `/opt/h2oai/dai/home/.driverlessai/license.sig`. If you are installing Driverless AI programmatically, you can copy a license key file to that location. If no license key is found, the application will interactively guide you to add one from the Web UI.
- systemd unit files are put in `/usr/lib/systemd/system`
- Symbolic links to the configuration files in `/etc/dai` files are put in `/etc/systemd/system`

If your environment is running an operational systemd, that is the preferred way to manage Driverless AI. The package installs the following systemd services and a wrapper service:

- **dai**: Wrapper service that starts/stops the other three services
Using Driverless AI, Release 1.6.2

- **dai-dai**: Main Driverless AI process
- **dai-h2o**: H2O-3 helper process used by Driverless AI
- **dai-procsy**: Procsy helper process used by Driverless AI
- **dai-vis-server**: Visualization server helper process used by Driverless AI

If you don’t have systemd, you can also use the provided run script to start Driverless AI.

### Starting NVIDIA Persistence Mode (GPU only)

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: [http://docs.nvidia.com/deploy/driver-persistence/index.html](http://docs.nvidia.com/deploy/driver-persistence/index.html).

```bash
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```

### Installing the Driverless AI DEB

Run the following commands to install the Driverless AI DEB. Replace VERSION with your specific version.

```
# Install Driverless AI.
sudo dpkg -i dai_VERSION.deb
```

By default, the Driverless AI processes are owned by the ‘dai’ user and ‘dai’ group. You can optionally specify a different service user and group as shown below. Replace `<myuser>` and `<mygroup>` as appropriate.

```
# Temporarily specify service user and group when installing Driverless AI.
# dpkg saves these for systemd in the /etc/dai/User.conf and /etc/dai/Group.conf files.
sudo DAI_USER=myuser DAI_GROUP=mygroup dpkg -i dai_VERSION.deb
```

You may now optionally make changes to `/etc/dai/config.toml`.

### Starting Driverless AI

If you have systemd (preferred):

```
# Start Driverless AI.
sudo systemctl start dai
```

If you do not have systemd:

```
# Start Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

### Looking at Driverless AI log files

If you have systemd (preferred):
Using Driverless AI, Release 1.6.2

Using Driverless AI

</syntaxhighlight>

If you do not have systemd:

```bash
sudo less /opt/h2oai/dai/log/dai.log
sudo less /opt/h2oai/dai/log/h2o.log
sudo less /opt/h2oai/dai/log/procsy.log
sudo less /opt/h2oai/dai/log/vis-server.log
```

**Stopping Driverless AI**

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai
```

**Upgrading Driverless AI**

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

The upgrade process inherits the service user and group from `/etc/dai/User.conf` and `/etc/dai/Group.conf`. You do not need to manually specify the DAI_USER or DAI_GROUP environment variables during an upgrade.

5.4. IBM Power Installs
Using Driverless AI, Release 1.6.2

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# Back up your /opt/h2oai/dai/tmp directory at this time. If you do not, all previous data will be lost.

# Upgrade Driverless AI.
sudo dpkg -i dai_NEWVERSION.deb
sudo systemctl daemon-reload
sudo systemctl start dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Upgrade and restart.
sudo dpkg -i dai_NEWVERSION.deb
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

Uninstalling Driverless AI

If you have systemd (preferred):

```bash
# Stop Driverless AI.
sudo systemctl stop dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall Driverless AI.
sudo dpkg -r dai

# Purge Driverless AI.
sudo dpkg -P dai
```

If you do not have systemd:

```bash
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Uninstall Driverless AI.
sudo dpkg -r dai

# Purge Driverless AI.
sudo dpkg -P dai
```
CAUTION! At this point you can optionally completely remove all remaining files, including the database. (This cannot be undone.)

```
sudo rm -rf /opt/h2oai/dai
sudo rm -rf /etc/dai
```

### 5.4.4 IBM TAR SH

The Driverless AI software is available for use in pure user-mode environments as a self-extracting TAR SH archive. This form of installation does not require a privileged user to install or to run.

This artifact has the same compatibility matrix as the RPM and DEB packages (combined), it just comes packaged slightly differently. See those sections for a full list of supported environments.

The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/products/h2o-driverless-ai/](https://www.h2o.ai/products/h2o-driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in.

**Note:** OpenCL and LightGBM with GPUs are not supported on Power currently.

**Requirements**

- RedHat 7 or Ubuntu 16.04
- Cuda 9 or later with NVIDIA drivers >= 396
- cuDNN >=7.2.1 (Required only if using TensorFlow.)
- Driverless AI TAR SH, available from [https://www.h2o.ai/download/](https://www.h2o.ai/download/)

**Installing Driverless AI**

Run the following commands to install the Driverless AI RPM. Replace VERSION with your specific version.

```
# Install Driverless AI.
chmod 755 dai-VERSION.sh
./dai-VERSION.sh
```

You may now cd to the unpacked directory and optionally make changes to `config.toml`.

**Starting Driverless AI**

```
# Start Driverless AI.
./run-dai.sh
```

**Starting NVIDIA Persistence Mode**

If you have NVIDIA GPUs, you must run the following two NVIDIA commands. These commands need to be run every reboot. For more information: [http://docs.nvidia.com/deploy/driver-persistence/index.html](http://docs.nvidia.com/deploy/driver-persistence/index.html).

```
sudo nvidia-persistenced --user dai
sudo nvidia-smi -pm 1
```
Looking at Driverless AI log files

```
less log/dai.log
less log/h2o.log
less log/procsy.log
less log/vis-server.log
```

Stopping Driverless AI

```
# Stop Driverless AI.
./kill-dai.sh
```

Uninstalling Driverless AI

To uninstall Driverless AI, just remove the directory created by the unpacking process. By default, all files for Driverless AI are contained within this directory.

Upgrading Driverless AI

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

1. Stop your previous version of Driverless AI.
2. Run the self-extracting archive for the new version of Driverless AI.
3. Port any previous changes you made to your config.toml file to the newly unpacked directory.
4. Copy the tmp directory (which contains all the Driverless AI working state) from your previous Driverless AI installation into the newly unpacked directory.
5. Start your newly extracted version of Driverless AI.

5.4.5 Troubleshooting IBM Installations

Opening Port 12345

For default IBM Power9 systems with RHEL 7 installed, be sure to open port 12345 in the firewall. For example:
firewall-cmd --zone=public --add-port=12345/tcp --permanent
firewall-cmd --reload

Growing the Disk

Some users may find it necessary to grow their disk. An example describing how to add disk space to a virtual machine is available at https://www.geoffstratton.com/expand-hard-disk-ubuntu-lvm. The steps for an IBM Power9 system with RHEL 7 would be similar.

5.5 Mac OS X

This section describes how to install, start, stop, and upgrade the Driverless AI Docker image on Mac OS X. Note that this uses regular Docker and not NVIDIA Docker. GPU support will not be available.

The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit https://www.h2o.ai/driverless-ai/. Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

Caution:

- This is an extremely memory-constrained environment for experimental purposes only. Stick to small datasets! For serious use, please use Linux.
- Be aware that there are known performance issues with Docker for Mac. More information is available here: https://docs.docker.com/docker-for-mac/osxfs/#technology.

5.5.1 Environment

<table>
<thead>
<tr>
<th>Operating System</th>
<th>GPU Support?</th>
<th>Min Mem</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X</td>
<td>No</td>
<td>16 GB</td>
<td>Experimentation</td>
</tr>
</tbody>
</table>

5.5.2 Installing Driverless AI

1. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.
2. Download and run Docker for Mac from https://docs.docker.com/docker-for-mac/install.
3. Adjust the amount of memory given to Docker to be at least 10 GB. Driverless AI won’t run at all with less than 10 GB of memory. You can optionally adjust the number of CPUs given to Docker. You will find the controls by clicking on (Docker Whale)->Preferences->Advanced as shown in the following screenshots. (Don’t forget to Apply the changes after setting the desired memory value.)
Chapter 5. Installing and Upgrading Driverless AI
4. Set up a directory for the version of Driverless AI within the Terminal, replacing VERSION below with your Driverless AI Docker image version:

```bash
mkdir dai_rel_VERSION
```

5. With Docker running, open a Terminal and move the downloaded Driverless AI to your new directory.

6. Change directories to the new directory, then load the image using the following command. This example shows how to load Driverless AI. Replace VERSION with your image.

```bash
cd dai_rel_VERSION
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

7. Set up the data, log, license, and tmp directories (within the new Driverless AI directory):

```bash
mkdir data
mkdir log
mkdir license
mkdir tmp
```

8. Optionally copy data into the `data` directory on the host. The data will be visible inside the Docker container at `/data`. You can also upload data after starting Driverless AI.

9. Run `docker images` to find the image tag.
10. Start the Driverless AI Docker image (still within the new Driverless AI directory). Replace TAG below with the image tag. Note that GPU support will not be available.

```
docker run \
  --pid=host \n  --init \n  --rm \n  --shm-size=256m \n  -u `id -u`:`id -g` \n  -p 12345:12345 \n  -v `pwd`/data:/data \n  -v `pwd`/log:/log \n  -v `pwd`/license:/license \n  -v `pwd`/tmp:/tmp \n  h2oai/dai-centos7-x86_64:TAG
```


### 5.5.3 Stopping the Docker Image

To stop the Driverless AI Docker image, type Ctrl + C in the Terminal (Mac OS X) or PowerShell (Windows 10) window that is running the Driverless AI Docker image.

### 5.5.4 Upgrading the Docker Image

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

**Note:** Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:

```
# Set up directory with the version name
mkdir dai_rel_VERSION

# cd into the new directory
cd dai_rel_VERSION
```

3. Retrieve the Driverless AI package from https://www.h2o.ai/download/ and add it to the new directory.
4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace VERSION with your image.

```
# Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:

```
# Copy the data, log, license, and tmp directories on the host machine
cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
```

At this point, your experiments from the previous versions will be visible inside the Docker container.

6. Use `docker images` to find the new image tag.

7. Start the Driverless AI Docker image.


### 5.6 Windows 10 Pro

This section describes how to install, start, stop, and upgrade Driverless AI on a Windows 10 Pro machine. The installation steps assume that you have a license key for Driverless AI. For information on how to obtain a license key for Driverless AI, visit [https://www.h2o.ai/driverless-ai/](https://www.h2o.ai/driverless-ai/). Once obtained, you will be prompted to paste the license key into the Driverless AI UI when you first log in, or you can save it as a .sig file and place it in the license folder that you will create during the installation process.

#### 5.6.1 Overview of Installation on Windows

The recommended way of installing Driverless AI on Windows is via WSL Ubuntu. Running a Driverless AI Docker image on Windows is also possible but not preferred.

**Note**: GPU support is not available on Windows.

**Caution**: This should be used only for experimental purposes and only on small data. For serious use, please use Linux.

#### 5.6.2 Environment

<table>
<thead>
<tr>
<th>Operating System</th>
<th>GPU Support?</th>
<th>Min Mem</th>
<th>Suitable for</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows 10 Pro</td>
<td>No</td>
<td>16 GB</td>
<td>Experimentation</td>
</tr>
</tbody>
</table>

#### 5.6.3 DEB Installs

This section describes how to install the Driverless AI DEB on Windows 10 using Windows Subsystem for Linux (WSL).
Requirements

- Ubuntu 18.04 from the Windows Store. (Note that Ubuntu 16.04 for WSL is no longer supported.)
- Driverless AI DEB, available from https://www.h2o.ai/download/.

Installation Procedure

(Note that systemd is not supported for Linux on Windows.)

Run the following commands to install and run the Driverless AI DEB. Replace <VERSION> with your specific version.

```
# Install Driverless AI. Expect installation of the .deb file to take several minutes on WSL.
sudo dpkg -i dai_VERSION.deb

# Run Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh
```

Upgrading the DEB

The Driverless AI Windows DEB cannot be upgraded. In order to run to a newer version, you must first uninstall the prior version and then install the newer one.

**WARNING:** Perform the following before uninstalling and then reinstalling Driverless AI.

- Build MLI models.
- Build MOJO pipelines.
- Back up your Driverless AI tmp directory.

If you did not build MLI on a model before installing a newer Driverless AI version, then you will not be able to view MLI on that model after. Before installing a newer version, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after installing the newer version.

If you did not build a MOJO pipeline on a model before installing a newer Driverless AI version then you will not be able to build a MOJO pipeline on that model after. Before installing a newer version, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

Run the following commands to uninstall a prior version.

```
# Stop Driverless AI.
sudo pkill -U dai

# The processes should now be stopped. Verify.
sudo ps -u dai

# Back up your /opt/h2oai/dai/tmp directory at this time.

# Uninstall Driverless AI.
sudo dpkg -r dai

# If the above uninstall command results in a message
# "failed to lookup unit file state: invalid argument,"
```
At this point, follow the previous installation procedure to install a newer version of Driverless AI.

## 5.6.4 Docker Image Installs

### Notes:

- Installing the Driverless AI Docker image on Windows is not the recommended method for running Driverless AI. RPM and DEB installs are preferred.
- Be aware that there are known issues with Docker for Windows. More information is available here: https://github.com/docker/for-win/issues/188.
- Consult with your Windows System Admin if
  - Your corporate environment does not allow third-party software installs
  - You are running Windows Defender
  - Your machine is not running with `Enable-WindowsOptionalFeature -Online -FeatureName Microsoft-Windows-Subsystem-Linux`.

**Watch the installation video** here. Note that some of the images in this video may change between releases, but the installation steps remain the same.

### Requirements

- Windows 10 Pro

### Installation Procedure

1. Retrieve the Driverless AI Docker image from https://www.h2o.ai/download/.
2. Download, install, and run Docker for Windows from https://docs.docker.com/docker-for-windows/install/. You can verify that Docker is running by typing `docker version` in a terminal (such as Windows PowerShell). Note that you may have to reboot after installation.
3. Before running Driverless AI, you must:
   - Enable shared access to the C drive. Driverless AI will not be able to see your local data if this is not set.
   - Adjust the amount of memory given to Docker to be at least 10 GB. Driverless AI won’t run at all with less than 10 GB of memory.
   - Optionally adjust the number of CPUs given to Docker.

You can adjust these settings by clicking on the Docker whale in your taskbar (look for hidden tasks, if necessary), then selecting **Settings > Shared Drive** and **Settings > Advanced** as shown in the following screenshots. Don’t forget to Apply the changes after setting the desired memory value. (Docker will restart.) Note that if you cannot make changes, stop Docker and then start Docker again by right clicking on the Docker icon on your desktop and selecting **Run as Administrator**.
Chapter 5. Installing and Upgrading Driverless AI
4. Open a PowerShell terminal and set up a directory for the version of Driverless AI on the host machine, replacing VERSION below with your Driverless AI Docker image version:

```bash
md dai_rel_VERSION
```

5. With Docker running, navigate to the location of your downloaded Driverless AI image. Move the downloaded Driverless AI image to your new directory.

6. Change directories to the new directory, then load the image using the following command. This example shows how to load Driverless AI. Replace VERSION with your image.

```bash
cd dai_rel_VERSION
docker load -i .\dai-docker-centos7-x86_64-VERSION.tar.gz
```

7. Set up the data, log, license, and tmp directories (within the new directory).

```bash
md data
md log
md license
md tmp
```

8. Copy data into the `/data` directory. The data will be visible inside the Docker container at `/data`.

9. Run `docker images` to find the image tag.

10. Start the Driverless AI Docker image. Be sure to replace `path_to_` below with the entire path to the location of the folders that you created (for example, “c:/Users/user-name/driverlessai_folder/data”), and replace `TAG` with the Docker image tag. Note that this is regular Docker, not NVIDIA Docker. GPU support will not be available.

```bash
docker run --pid=host --init --rm --shm-size=256m -p 12345:12345 -v c:/path_to_data:/data -v c:/path_to_log:/log -v c:/path_to_license:/license -v c:/path_to_tmp:/tmp h2oai/dai-centos7-x86_64:TAG
```

Stopping the Docker Image

To stop the Driverless AI Docker image, type `Ctrl + C` in the Terminal (Mac OS X) or PowerShell (Windows 10) window that is running the Driverless AI Docker image.

Upgrading the Docker Image

This section provides instructions for upgrading Driverless AI versions that were installed in a Docker container. These steps ensure that existing experiments are saved.

**WARNING:** Experiments, MLIs, and MOJOs reside in the Driverless AI tmp directory and are not automatically upgraded when Driverless AI is upgraded.

- Build MLI models before upgrading.
- Build MOJO pipelines before upgrading.
- Back up your Driverless AI tmp directory before upgrading.

If you did not build MLI on a model before upgrading Driverless AI, then you will not be able to view MLI on that model after upgrading. Before upgrading, be sure to run MLI jobs on models that you want to continue to interpret in future releases. If that MLI job appears in the list of Interpreted Models in your current version, then it will be retained after upgrading.

If you did not build a MOJO pipeline on a model before upgrading Driverless AI, then you will not be able to build a MOJO pipeline on that model after upgrading. Before upgrading, be sure to build MOJO pipelines on all desired models and then back up your Driverless AI tmp directory.

**Note:** Stop Driverless AI if it is still running.

1. SSH into the IP address of the machine that is running Driverless AI.
2. Set up a directory for the version of Driverless AI on the host machine:

```
# Set up directory with the version name
mkdir dai_rel_VERSION
# cd into the new directory
cd dai_rel_VERSION
```

3. Retrieve the Driverless AI package from https://www.h2o.ai/download/ and add it to the new directory.

4. Load the Driverless AI Docker image inside the new directory. This example shows how to load Driverless AI version. If necessary, replace VERSION with your image.

```
# Load the Driverless AI docker image
docker load < dai-docker-centos7-x86_64-VERSION.tar.gz
```

5. Copy the data, log, license, and tmp directories from the previous Driverless AI directory to the new Driverless AI directory:

```
# Copy the data, log, license, and tmp directories on the host machine
cp -r dai_rel_1.4.2/data dai_rel_VERSION/data
cp -r dai_rel_1.4.2/log dai_rel_VERSION/log
cp -r dai_rel_1.4.2/license dai_rel_VERSION/license
cp -r dai_rel_1.4.2/tmp dai_rel_VERSION/tmp
```

At this point, your experiments from the previous versions will be visible inside the Docker container.

6. Use `docker images` to find the new image tag.
7. Start the Driverless AI Docker image.

Admins can edit a config.toml file when starting the Driverless AI Docker image. The config.toml file includes all possible configuration options that would otherwise be specified in the `nvidia-docker run` command. This file is located in a folder on the container. You can make updates to environment variables directly in this file. Driverless AI will use the updated config.toml file when starting from native installs. Docker users can specify that updated config.toml file when starting Driverless AI Docker image.

### 6.1 Docker Image Users

1. Copy the config.toml file from inside the Docker image to your local filesystem.

   ```bash
   # Make a config directory
   mkdir config

   # Copy the config.toml file to the new config directory.
   nvidia-docker run \
   --pid=host \n   --rm \n   --init \n   -u `id -u`:`id -g` \n   -v `pwd`/config:/config \n   --entrypoint bash \n   h2oai/dai-centos7-x86_64:TAG \n   -c "cp /etc/dai/config.toml /config"
   ```

2. Edit the desired variables in the config.toml file. Save your changes when you are done.

3. Start Driverless AI with the DRIVERLESS_AL_CONFIG_FILE environment variable. Make sure this points to the location of the edited config.toml file so that the software finds the configuration file.

   ```bash
   nvidia-docker run \
   --pid=host \n   --init \n   --rm \n   --shm-size=256m \n   -u `id -u`:`id -g` \n   -p 12345:12345 \n   -e DRIVERLESS_AL_CONFIG_FILE="/config/config.toml" \n   -v `pwd`/config:/config \n   -v `pwd`/data:/data \n   -v `pwd`/log:/log \n   -v `pwd`/license:/license 
   ```

(continues on next page)
6.2 Native Install Users

Native installs include DEBs, RPMs, and TAR SH installs.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

   ```bash
   export DRIVERLESS_AI_CONFIG_FILE="/config/config.toml"
   ```

2. Edit the desired variables in the config.toml file. Save your changes when you are done.

3. Start Driverless AI. Note that the command used to start Driverless AI varies depending on your install type.

For reference, below is a copy of the standard config.toml file included with this version of Driverless AI. The sections that follow describe some examples showing how to set different environment variables, data connectors, authentication, and notifications.

6.3 Sample Config.toml File

```toml
[internal not visible to users]  
[place file in a folder/mount file in docker container and provide path in "DRIVERLESS_AI_CONFIG_FILE" environment variable]  
[configuration variables can also be provided as environment variables they must have the prefix "DRIVERLESS_AI_" followed by variable name in caps e.g "authentication_method" can be provided as "DRIVERLESS_AI_AUTHENTICATION_METHOD"]  
[Note: All floating point values < 1.0 need to start with 0. E.g. max_relative_cardinality = 0.95]  

Toml Control : Ways to control how toml parameters are set

# Whether to allow user to change non-server toml parameters per experiment in expert page
#allow_config_overrides_in_expert_page = true

# Instructions for 'Add to config.toml via toml string' in GUI expert page
# Self-referential toml parameter, for setting any other toml parameters as string of tomls
# separated by \n (spaces around \n are ok).
# Useful when toml parameter is not in expert mode but want per-experiment control.
```
Setting this will override all other choices.
# In expert page, each time expert options saved, the new state is set without memory of any
# prior settings.
# The entered item is a fully compliant toml string that would be processed directly by toml.
# load().
# One should include 2 double quotes around the entire setting, or double quotes need to be escaped.
# One enters into the expert page text as follows:
# e.g. enable_glm="off" \n enable_xgboost="off" \n enable_lightgbm="on"
# e.g. "enable_glm=off" \n enable_xgboost=off \n enable_lightgbm=off \n enable_tensorflow=on"
# e.g. fixed_num_individuals=4
# e.g. params_lightgbm="{'objective': 'poisson'}"
# e.g. "params_lightgbm="{'objective': 'poisson'}"
# e.g. max_cores=10 \n data_precision="float32" \n max_rows_feature_evolution=50000000000 \n ensemble_accuracy_switch=11 \n feature_engineering_effort=1 \n target_transformer="identity"
# ensemble_accuracy_switch=11 \n feature_engineering_effort=1 \n target_transformer="identity"
# max_cores=10 \n data_precision="float32" \n max_rows_feature_evolution=50000000000 \n ensemble_accuracy_switch=11 \n feature_engineering_effort=1 \n target_transformer="identity"
# e.g. max_cores=10 \n data_precision="float32" \n max_rows_feature_evolution=50000000000 \n ensemble_accuracy_switch=11 \n feature_engineering_effort=1 \n target_transformer="identity"
# If you see: "toml.TomlDecodeError" then ensure toml is set correctly.
# When set in the expert page of an experiment, these changes only affect experiments and not the server
# Usually should keep this as empty string in this toml file.
#configOverrides = ''

# Setup : Configure application server here (ip, ports, authentication, file types etc)

# IP address and port of autoviz process.
# vis_server_ip = "127.0.0.1"
# vis_server_port = 12346

# IP address and port of procsy process.
# procsy_ip = "127.0.0.1"
# procsy_port = 12347

# IP address and port of H2O instance.
# h2o_ip = "127.0.0.1"
# h2o_port = 54321

# IP address and port for Driverless AI HTTP server.
# ip = "127.0.0.1"
# port = 12345

# File upload limit (default 100GB)
# max_file_upload_size = 104857600000

# Verbosity of logging
# 0: quiet (CRITICAL, ERROR, WARNING)
# 1: default (CRITICAL, ERROR, WARNING, INFO, DATA)
# 2: verbose (CRITICAL, ERROR, WARNING, INFO, DATA, DEBUG)
# Affects server and all experiments
# log_level = 1

# Whether to collect relevant server logs (h2oai_server.log, dai.log from systemctl or docker,
# and h2o log)
# Useful for when sending logs to H2O.ai
#collect_server_logs_in_experiment_logs = false

# Redis
#redis_ip = "127.0.0.1"
#redis_port = 6379
# Using Driverless AI, Release 1.6.2

(continued from previous page)

```toml
#master Redis password = ""
#
https settings
#
# You can make a self-signed certificate for testing with the following commands:
#
# sudo openssl req -x509 -newkey rsa:4096 -keyout private_key.pem -out cert.pem -days 3650 -
# -nodes -subj "/O=Driverless AI"
# sudo chown dai:dai cert.pem private_key.pem
# sudo chmod 600 cert.pem private_key.pem
# sudo mv cert.pem private_key.pem /etc/dai
#
#enable https = false
#ssl_key_file = "/etc/da/private_key.pem"
#ssl_crt_file = "/etc/da/cert.pem"
#
SSL TLS
#ssl_no_sslv2 = true
#ssl_no_sslv3 = true
#ssl_no_tlsv1 = true
#ssl_no_tlsv1_1 = true
#ssl_no_tlsv1_2 = false
#ssl_no_tlsv1_3 = false
#
# Data directory. All application data and files related datasets and
# experiments are stored in this directory.
#data_directory = ".//tmp"
#
# Whether to run quick performance benchmark at start of application
#enable_benchmark = true
#
# Whether to run quick startup checks at start of application
#enable_startup_checks = true
#
# Whether to opt in to usage statistics and bug reporting
#usage_stats_opt_in = false
#
# authentication_method
#unvalidated : Accepts user id and password. Does not validate password.
#none: Does not ask for user id or password. Authenticated as admin.
#openid: Users OpenID Connect provider for authentication. See additional OpenID settings below.
#pam: Accepts user id and password. Validates user with operating system.
#ldap: Accepts user id and password. Validates against an ldap server. Look
# for additional settings under LDAP settings.
#local: Accepts a user id and password. Validated against an htpasswd file provided in local_—htpasswd_file.
# ibm_spectrum_conductor: Authenticate with IBM conductor auth api.
#authentication_method = "unvalidated"
#
default amount of time in hours before we force user to login again (if not provided by—
—authentication_method)
#authentication_default_timeout_hours = 72
#
# OpenID Connect Settings:
# base server uri to the OpenID Provider server (ex: http://localhost:7777)
#auth_openid_provider_base_url=""
# uri to pull OpenID config data from (you can extract most of required OpenID config from this_—url)
# usually located at: /auth/realms/master/.well-known/openid-configuration
#auth_openid_configuration_uri=""
# uri to start authentication flow
#auth_openid_auth_uri=""
# uri to make request for token after callback from OpenID server was received
#auth_openid_token_uri=""
# uri to get user information once access_token has been acquired (ex: list of groups user_
—belongs to will be provided here)
```

(continues on next page)
# auth_openid_userinfo_uri=
# uri to logout user
# auth_openid_logout_uri=
# callback uri that OpenID provider will use to send "authentication_code"
# auth_openid_redirect_uri=
# OAuth2 grant type (usually acceess_token)
# auth_openid_grant_type=
# OAuth2 response type (usually code)
# auth_openid_response_type=
# Client ID registered with OpenID provider
# auth_openid_client_id=
# Client secret provided by OpenID provider when registering Client ID
# auth_openid_client_secret=
# Scope of info (usually openid)
# auth_openid_scope=
# What key in user_info json should we check to authorize user
# auth_openiduserinfo_auth_key=
# What value should the key have in user_info json in order to authorize user
# auth_openiduserinfo_auth_value=
# Key that specifies username in user_info json (we will use it as username in our system)
# auth_openiduserinfo_username_key=

# server_cookie_expiration_days
# Sets the time until expiration of secure cookie issued by server to client
# Cookie is issued upon login to Driverless AI UI and will expire 'n' days after that point
# If you wish cookies to expire in less than 1 day use decimals (1 day / 24 hours = 0.042 day/
# hour --> cookie expires in 1 hour)
# Expected behavior: if user is logged in, and cookie expires, the next click will redirect the
# user to the login page of the Driverless AI UI.
#server_cookie_expiration_days = 30

# LDAP Configuration
#ldap_server = # ldap server domain or ip
#ldap_port = # ldap server port
#ldap_bind_dn = # Complete DN of the LDAP bind user
#ldap_bind_password = #Password for the LDAP bind
#ldap_tls_file = # Provide Cert file location
#ldap_use_ssl = # use true to use ssl or false
#ldap_search_base = # the location in the DIT where the search will start
#ldap_search_filter = # a string that describes what you are searching for
#ldap_search_attributes = # ldap attributes to return from search
#ldap_user_name_attribute = "uid" # specify key to find user name

# LDAP depreicated settings
#ldap_recipe = "0" # When using this recipe, needs to be set to "1"
#ldap_user_prefix = # Deprecated do not use
#ldap_search_user_id = # Deprecated, Use ldap_bind_dn
#ldap_search_password = # Deprecated, ldap_bind_password
#ldap_ou_dn = # Deprecated, use ldap_search_base instead
#ldap_do = # Deprecated, use ldap_base_dn
#ldap_base_dn = # Deprecated, use ldap_search_base
#ldap_base_filter = # Deprecated use ldap_search_filter

# Local password file
# Generating a htpasswd file: see syntax below
# htpasswd -B "<location_to_place_htpasswd_file>" "<username>"
# note: -B forces use of brcypt, a secure encryption method
#local_htpasswd_file = 

# Supported file formats (file name endings must match for files to show up in file browser)
supported_file_types = "csv, tsv, txt, dat, tgz, gz, bz2, zip, xz, xls, xlsx, nff, jay, feather,
# bin, arff, parquet"

# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs\_path\_to\_service\_account\_json below
# gbq : Google Big Query, remember to configure gcs\_path\_to\_service\_account\_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account, name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm\_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

#enabled\_file\_systems = "upload, file, hdfs, s3"

# do\_not\_log\_list : add configurations that you do not wish to be recorded in logs here

#do\_not\_log\_list = "local\_htpasswd\_file, aws\_access\_key\_id, aws\_secret\_access\_key, snowflake\_password, snowflake\_url, snowflake\_user, snowflake\_account, minio\_endpoint\_url, minio\_access\_key\_id, minio\_secret\_access\_key, kdb\_user, kdb\_password, ldap\_bind\_password, gcs\_path\_to\_service\_account\_json, azure\_blob\_account\_name, azure\_blob\_account\_key, deployment\_aws\_access\_key\_id, deployment\_aws\_secret\_access\_key, master\_minio\_access\_key\_id, master\_minio\_secret\_access\_key, master\_redis\_password, auth\_openid\_client\_id, auth\_openid\_client\_secret, auth\_openid\_userinfo\_auth\_key, auth\_openid\_userinfo\_auth\_value, auth\_openid\_userinfo\_username\_key"

# Minio is used for file distribution on multinode architecture
# These settings are used to specify the local Minio connection to master nodes
#master\_minio\_address = "<URL>:<PORT>"
#master\_minio\_access\_key\_id = ""
#master\_minio\_secret\_access\_key = ""

#allow\_local\_storage = true

### Scoring Artifacts: Setup which scoring artifacts to generate by default
# Whether to create the Python scoring pipeline at the end of each experiment
#make\_python\_scoring\_pipeline = true

# Whether to create the MOJO scoring pipeline at the end of each experiment
# Note: Not all transformers or main models are available for MOJO (e.g. no gblinear main model)
#make\_mojo\_scoring\_pipeline = false

### Hardware: Configure hardware settings here (GPUs, CPUs, Memory, etc.)
# Max number of CPU cores to use per experiment. Set to <= 0 to use all cores.
# One can also set environment variable "OMP\_NUM\_THREADS" to number of cores to use for OpenMP
# e.g. In bash: export OMP\_NUM\_THREADS=32 and export OPENBLAS\_NUM\_THREADS=32
# Set to -1 for all available cores.
#max\_cores = -1

# Whether to set automatic number of cores by physical (true) or logical (false) count
# Using all logical cores can lead to poor performance due to cache thrashing
#max\_cores\_by\_physical = true

# Absolute limit to core count
#max\_cores\_limit = 100

# Number of GPUs to use per experiment for training task. Set to -1 for all GPUs.
# An experiment will generate many different models.
# Currently num\_gpus\_per\_experiment!=-1 disables GPU locking, so is only recommended for single experiments and single users.
# Ignored if GPUs disabled or no GPUs on system.
# More info at: https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker#gpu-isolation
#num\_gpus\_per\_experiment = -1
# Number of GPUs to use per model training task. Set to -1 for all GPUs.
# For example, when this is set to -1 and there are 4 GPUs available, all of them can be used.
# for the training of a single model.
# Currently num_gpus_per_model!=1 disables GPU locking, so is only recommended for single
# experiments and single users.
# Ignored if GPUs disabled or no GPUs on system.
# More info at: https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker#gpu-isolation
#num_gpus_per_model = 1

# Minimum number of threads for datatable (and OpenMP) during data munging
data_table is the main data munging tool used within Driverless ai (source: https://github.com/h2oai/datatable)
#min_dt_threads_munging = 4

# Like min_datatable (and OpenMP)_threads_munging but for final pipeline munging
#min_dt_threads_final_munging = 4

# Which gpu_id to start with
# If using CUDA_VISIBLE_DEVICES=... to control GPUs (preferred method), gpu_id=0 is the
# first in that restricted list of devices.
# E.g. if CUDA_VISIBLE_DEVICES="4,5" then gpu_id_start=0 will refer to the
device 4.
# E.g. from expert mode, to run 2 experiments, each on a distinct GPU out of 2 GPUs:
# Experiment#1: num_gpus_per_model=1, num_gpus_per_experiment=1, gpu_id_start=0
# Experiment#2: num_gpus_per_model=1, num_gpus_per_experiment=1, gpu_id_start=1
# E.g. from expert mode, to run 2 experiments, each on a distinct GPU out of 8 GPUs:
# Experiment#1: num_gpus_per_model=1, num_gpus_per_experiment=4, gpu_id_start=0
# Experiment#2: num_gpus_per_model=1, num_gpus_per_experiment=4, gpu_id_start=4
# E.g. Like just above, but now run on all 4 GPUs/model
# Experiment#1: num_gpus_per_model=4, num_gpus_per_experiment=4, gpu_id_start=0
# Experiment#2: num_gpus_per_model=4, num_gpus_per_experiment=4, gpu_id_start=4
# If num_gpus_per_model=1, global GPU locking is disabled
# (because underlying algorithms don't support arbitrary gpu ids, only sequential ids),
# so must setup above correctly to avoid overlap across all experiments by all users
# More info at: https://github.com/NVIDIA/nvidia-docker/wiki/nvidia-docker#gpu-isolation
# Note that gpu selection does not wrap, so gpu_id_start + num_gpus_per_model must be less than
# number of visible gpus
#gpu_id_start = 0

# Maximum number of workers for DriverlessAI server pool (only 1 needed
# currently)
#max_workers = 1

# Period (in seconds) of ping by DriverlessAI server to each experiment
# (in order to get logger info like disk space and memory usage)
# 0 means don't print anything
#ping_period = 60

# Minimum amount of disk space in GB needed to run experiments.
# Experiments will fail if this limit is crossed.
# This limit exists, because Driverless AI needs to generate data for model training
# feature engineering, documentation and other such processes.
# Disk space can also be used as an alternative to system memory (RAM).
disk_limit_gb = 5

# Minimum amount of system memory in GB needed to start experiments
# Similarly with disk space, a certain amount of system memory is needed to run some basic
# operations.
#memory_limit_gb = 5

# Minimum number of rows needed to run experiments (values lower than 100
# might not work)
# A minimum threshold is set to ensure there is enough data to create a statistically
# reliable model and avoid other small-data related failures.
#min_num_rows = 100
Using Driverless AI, Release 1.6.2

(continued from previous page)

# Minimum required number of rows (in the training data) for each class label for
classification problems.
#min_rows_per_class = 5

# Minimum required number of rows for each split when generating validation samples.
#min_rows_per_split = 5

# Precision of how data is stored
# "float32" best for speed, "float64" best for accuracy or very large input values
# "float32" allows numbers up to about +3E38 with relative error of about 1E-7
# "float64" allows numbers up to about +1E308 with relative error of about 1E-16
# Some calculations, like the GLM standardization, can only handle up to sqrt() of these
#-maximums for data values,
# So GLM with 32-bit precision can only handle up to a value of 1E19 before
#-standardization generates inf values.
# If you see "Best individual has invalid score" you may require higher precision.
#data_precision = "float32"

# Precision of most data transformers
# (Same options and notes as data_precision)
# Useful for higher precision in transformers with numerous operations that can accumulate error
# Also useful if want faster performance for transformers but otherwise want data stored in high
#-precision
#transformer_precision = "float32"

# Whether to change ulimit soft limits up to hard limits (for DAI server app, which is not a
#-generic user app)
# Prevents resource limit problems in some cases
# Restricted to no more than limit_nofile and limit_nproc for those resources
#ulimit_up_to_hard_limit = true

# number of file limit
# Below should be consistent with start-dai.sh
#limit_nofile=65535

# number of threads limit
# Below should be consistent with start-dai.sh
#limit_nproc=16384

##############################################################################
## Machine Learning: Configure machine learning configurations here
# (Data, Feature Engineering, Modelling etc)

# Seed for random number generator to make experiments reproducible (on same hardware), only
#-active if 'reproducible' mode is enabled
#seed = 1234

# List of values that should be interpreted as missing values during data import. Applies both
#-to numeric and string columns. Note that 'nan' is always interpreted as a missing value for
#-numeric columns.
#missing_values = "['', '?', 'None', 'nan', 'NA', 'N/A', 'unknown', 'inf', '-inf', '1.
#-7976931348623157e+308', '-1.7976931348623157e+308']"

# For tensorflow, what numerical value to give to missing values, where numeric values are
#-standardized
# So 0 is center of distribution, and if Normal distribution then +-5 is 5 standard deviations
#-away from the center.
# In many cases, an out of bounds value is a good way to represent missings, but in some cases
#-the mean (0) may be better.
#tf_nan_impute_value = -5

# Internal threshold for number of rows x number of columns to trigger certain statistical
# techniques (small data recipe like including one hot encoding for all model types, and smaller
#-learning rate)
# to increase model accuracy

(continues on next page)
Using Driverless AI, Release 1.6.2

---

#statistical_threshold_data_size_small = 100000

# Internal threshold for number of rows x number of columns to trigger certain statistical
techniques (fewer genes created, removal of high max_depth for tree models, etc.) that can
→ speed up modeling
# Also controls maximum rows used in training final model,
# by sampling statistical_threshold_data_size_large / columns number of rows
#statistical_threshold_data_size_large = 100000000

# Internal threshold for number of rows x number of columns to trigger sampling for auxiliary
→ data uses,
# like imbalanced data set detection and bootstrap scoring sample size and iterations
#aux_threshold_data_size_large = 10000000

# Internal threshold for number of rows x number of columns to trigger certain changes in
→ performance
# (fewer threads if beyond large value) to help avoid OOM or unnecessary slowdowns
# (fewer threads if lower than small value) to avoid excess forking of tasks
#performance_threshold_data_size_small = 100000
#performance_threshold_data_size_large = 100000000

# Upper limit on the number of rows x number of columns for feature evolution (applies to both
→ training and validation/holdout splits)
# feature evolution is the process that determines which features will be derived
# Depending on accuracy settings, a fraction of this value will be used
#feature_evolution_data_size = 100000000

# Maximum number of columns to start an experiment. This threshold exists to constraint the
→ complexity and the length of the Driverless AI's processes.
#max_cols = 10000

# Largest number of rows to use for column stats, otherwise sample randomly
#max_rows_col_stats = 1000000

# Maximum number of columns selected out of original set of original columns, using feature
→ selection
# The selection is based upon how well target encoding (or frequency encoding if not available)
→ on categoricals and numerics treated as categoricals
# This is useful to reduce the final model complexity. First the best
# [max_orig_cols_selected] are found through feature selection methods and then
# these features are used in feature evolution (to derive other features) and in modelling.
#max_orig_cols_selected = 10000

# Maximum number of numeric columns selected, above which will do feature selection
# same as above (max_orig_numeric_cols_selected) but for numeric columns.
#max_orig_numeric_cols_selected = 10000

# Maximum number of non-numeric columns selected, above which will do feature selection on all
→ features and avoid num_as_cat
# same as above (max_orig_numeric_cols_selected) but for categorical columns.
#max_orig_nonnumeric_cols_selected = 500

# The factor times max_orig_cols_selected, by which column selection is based upon no target
→ encoding and no num_as_cat
# in order to limit performance cost of feature engineering
#max_orig_cols_selected_simple_factor = 2

# Maximum allowed fraction of unique values for integer and categorical columns (otherwise will
→ treat column as ID and drop)
#max_relative_cardinality = 0.95

# Maximum allowed number of unique values for integer and categorical columns (otherwise will
→ treat column as ID and drop)
#max_absolute_cardinality = 1000000

# Whether to treat some numerical features as categorical

---

6.3. Sample Config.toml File 125
For instance, sometimes an integer column may not represent a numerical feature but represent different numerical codes instead.

- `num_as_cat = true`

- Max number of unique values for integer/real columns to be treated as categoricals (test_-applies to first statistical_threshold_data_size_small rows only)
  - `max_int_as_cat_uniques = 50`

- Number of folds for models used during the feature engineering process
  - Increasing this will put a lower fraction of data into validation and more into training
  - E.g. `num_folds=3` means 67%/33% training/validation splits
  - Actual value will vary for small or big data cases
  - `num_folds = 3`

- Accuracy setting equal and above which enables full cross-validation (multiple folds) during
  - Feature evolution
  - As opposed to only a single holdout split (e.g. 2/3 train and 1/3 validation holdout)
  - `full_cv_accuracy_switch = 8`

- Accuracy setting equal and above which enables stacked ensemble as final model
  - Stacking commences at the end of the feature evolution process.
  - It quite often leads to better model performance, but it does increase the complexity
  - And execution time of the final model
  - `ensemble_accuracy_switch = 5`

- Fixed ensemble level
  - `-1` - auto, based upon `ensemble_accuracy_switch`, accuracy, size of data, etc.
  - `0` - no ensemble, only final single model on validated iteration/tree count
  - `1` - 1 model, multiple ensemble folds (cross-validation)
  - `2` - 2 models, multiple ensemble folds (cross-validation)
  - `3` - 3 models, multiple ensemble folds (cross-validation)
  - `4` - 4 models, multiple ensemble folds (cross-validation)
  - `fixed_ensemble_level = -1`

- Number of fold splits to use for `ensemble_level >= 2`
  - The ensemble modeling may require predictions to be made on out-of-fold samples
  - Hence the data needs to be split on different folds to generate these predictions.
  - Less folds (like 2 or 3) normally create more stable models, but may be less accurate
  - More folds can get to higher accuracy at the expense of more time, but the performance
  - May be less stable when the training data is not enough (i.e. higher chance of overfitting).
  - Actual value will vary for small or big data cases
  - `num_ensemble_folds = 5`

- Number of repeats for each fold for all validation
  - (modified slightly for small or big data cases)
  - `fold_reps = 1`

- For binary classification: ratio of majority to minority class equal and above which to enable
  - undersampling
  - This option helps to deal with imbalance (on the target variable)
  - `imbalance_ratio_undersampling_threshold = 5`

- Quantile-based sampling method for imbalanced binary classification (only if class ratio is
  - above the threshold provided above)
  - Model on data is used to create deciles of predictions, and then each decile is sampled from
  - uniformly.
  - `quantile_imbalanced_sampling = false`

- Maximum number of classes to allow for a multi-classification problem.
  - High number of classes may make certain processes of Driverless AI time-consuming.
  - Memory requirements also increase with higher number of classes
  - `max_num_classes = 100`

- Number of actuals vs. predicted data points to use in order to generate in the relevant
  - plot/graph which is shown at the right part of the screen within an experiment.
  - `num_actuals_vs_predicted = 100`
Whether to use H2O.ai brain, the local caching and smart re-use of prior models to generate new features for new models.

This variable essentially controls how much information we store about the different models generated and different features explored while running an experiment. It can help with checkpointing and retrieving experiments that have been paused or interrupted.

Will use H2O.ai brain cache if cache file has no extra column names per column type, cache exactly matches classes, class labels, and time series options, interpretability of cache is equal or lower, main model (booster) is allowed by new experiment.

Level of brain to use (for chosen level, where higher levels will also do all lower levels): operations automatically

-1 = Don't use any brain cache and don't write any cache
0 = Don't use any brain cache but still write cache
Use case: Want to save model for later use, but want current model to be built without any brain models
1 = smart checkpoint if passed in old experiment_id to pull from (via GUI, running "restart from checkpoint" or chose which experiment to resume from)
Use case: From GUI select prior experiment using the right-hand panel, and select "RESTART FROM LAST CHECKPOINT" to use specific experiment's model to build new models from
2 = smart checkpoint from H2O.ai brain cache of individual best models
Use case: No need to select a particular prior experiment, we scan through H2O.ai brain cache for best models to restart from
3 = smart checkpoint like level #1, but for entire population. Tune only if brain population is insufficient size
(will re-score entire population in single iteration, so appears to take longer to complete first iteration)
4 = smart checkpoint like level #2, but for entire population. Tune only if brain population is insufficient size
(will re-score entire population in single iteration, so appears to take longer to complete first iteration)
5 = like #4, but will scan over entire brain cache of populations to get best scored individuals, starting from resumed experiment if chosen.
(can be slower due to brain cache scanning if big cache)
Other use cases:
a) Restart on different data: Use same column names and fewer or more rows (applicable to 1 - 5)
b) Re-fit only final pipeline: Like (a), but choose time=1 and feature_brain_level=3 - 5
c) Restart with more columns: Add columns, so model builds upon old model built from old column names (1 - 5)
d) Restart with focus on model tuning: Restart, then select feature_engineering_effort = 3 in expert settings
Notes:
1) For Restart cases, may want to set min_dai_iterations to non-zero to force delayed early stopping, else may not be enough iterations to find better model.
2) A "Restart from last checkpoint" of a Re-fit will fail to find cache and re-start fresh experiment
3) A "New model with Same Params" of a Re-fit will fail to find cache and re-start fresh experiment
4) A "New model with Same Params" of a Restart will use feature_brain_level=3 for default restart mode (revert to 2, or even 0 if want to start a fresh experiment otherwise)

# Maximum number of brain individuals pulled from H2O.ai brain cache for feature_brain_level=1, 2
max_num_brain_indivs = 3

# Directory, relative to data_directory, to store H2O.ai brain meta model files
brain_rel_dir = "H2O.ai_brain"

# Maximum size in bytes the brain will store
We reserve this memory to save data in order to ensure we can retrieve an experiment if for any reason it gets interrupted.
-1: unlimited
>=0 number of GB to limit brain to
brain_max_size_GB = 20
Whether to enable early stopping
# Early stopping refers to stopping the feature evolution/engineering process
when there is no performance uplift after a certain number of iterations.
# After early stopping has been triggered, Driverless AI will initiate the ensemble
# process if selected.
# early_stopping = true

# Minimum number of Driverless AI iterations to stop the feature evolution/engineering
# process even if score is not improving. Driverless AI needs to run for at least that many
# iterations before deciding to stop. It can be seen a safeguard against suboptimal (early)
# convergence.
# min_dai_iterations = 0

# Maximum features per model (and each model within the final model if ensemble) kept just after,
# scoring them.
# Final ensemble will exclude any pruned-away features and only train on kept features,
# but may contain a few new features due to fitting on different data view (e.g. new clusters)
# Final scoring pipeline will exclude any pruned-away features,
# but may contain a few new features due to fitting on different data view (e.g. new clusters)
# -1 means no restrictions except internally-determined memory restrictions
# nfeatures_max = -1

# Recipe type
# Recipes override any GUI settings
# 'auto' : all models and features automatically determined by experiment settings, toml
# settings, and feature_engineering_effort
# 'compliant' : like 'auto' except:
# 1 interpretabiliy=10 (to avoid complexity, overrides GUI or python client chose for
# --interpretability)
# 2 enable_glm='on' (rest 'off', to avoid complexity and be compatible with algorithms supported
# by MLI)
# 3 num_as_cat=false: don't convert any numerics to categoricals except via one-hot encoding (to
# --avoid complexity)
# 4 fixed_ensemble_level=0: Don't use any ensemble (to avoid complexity)
# 5 feature_brain_level=0: No feature brain used (to ensure every restart is identical)
# 6 max_feature_interaction_depth=1: interaction depth is set to 1 (no multi-feature interactions
# --to avoid complexity)
# 7 target_transformer='identity': for regression (to avoid complexity)
# 8 check_distribution_shift=false: Don't use distribution shift between train, valid, and test to
# --drop features (bit risky without fine-tuning)
# recipe = 'auto'

# How much effort to spend on feature engineering (0...10)
# Heuristic combination of various developer-level toml parameters
# 0 : keep only numeric features, only model tuning during evolution
# 1 : keep only numeric features and frequency-encoded categoricals, only model tuning during
# evolution
# 2 : Like #1 but instead just no Text features. Some feature tuning before evolution.
# 3 : Like #5 but only tuning during evolution. Mixed tuning of features and model parameters.
# 4 : Like #5, but slightly more focused on model tuning
# 5 : Default. Balanced feature-model tuning
# 6-7 : Like #5, but slightly more focused on feature engineering
# 8 : Like #6-7, but even more focused on feature engineering with high feature generation
# --rate, no feature dropping even if high interpretability
# 9-10: Like #8, but no model tuning during feature evolution
# feature_engineering_effort = 5

# Threshold for average string-is-text score as determined by internal heuristics
# It decides when a string column will be treated as text (for an NLP problem) or just as
# a standard categorical variable.
# Higher values will favor string columns as categoricals, lower values will favor string
# --columns as text
# string_col_as_text_threshold = 0.3
# Minimum fraction of unique values for string columns to be considered as possible text → (otherwise categorical)
#string_col_as_text_min_relative_cardinality = 0.1

# Minimum number of uniques for string columns to be considered as possible text (otherwise → categorical)
#string_col_as_text_min_absolute_cardinality = 100

# Interpretability setting equal and above which will use monotonicity constraints in GBM
# You may read the following source to understand what these constraints connote and why
# they may be important, especially when the end goal is a very interpretable machine
# learning model: https://blog.datadive.net/monotonicity-constraints-in-machine-learning/
#monotonicity_constraints_interpretability_switch = 7

# Exploring feature interactions can be important in gaining better predictive performance.
# The interaction can take multiple forms (i.e. feature1 + feature2 or feature1 * feature2 + ... → featureN)
# Although certain machine learning algorithms (like tree-based methods) can do well in
# capturing these interactions as part of their training process, still generating them may
# help them (or other algorithms) yield better performance.
# The depth of the interaction level (as in "up to" how many features may be combined at
# once to create one single feature) can be specified to control the complexity of the
# feature engineering process. Higher values might be able to make more predictive models
# at the expense of time.
#max_feature_interaction_depth = 8

# Accuracy setting equal and above which enables tuning of model parameters
# Only applicable if parameter_tuning_num_models=-1 (auto)
tune_parameters_accuracy_switch = 3

# Number of models to tune during pre-evolution phase
# Can make this lower to avoid excessive tuning, or make higher to do enhanced tuning
#-1 : auto
#parameter_tuning_num_models = -1

# Accuracy setting equal and above which enables tuning of target transform for regression
# This is useful for time series when instead of predicting the actual target value, it
# might be better to predict a transformed target variable like sqrt(target) or log(target)
# as a means to control for outliers.
tune_target_transform_accuracy_switch = 3

# Whether to automatically select target transformation for regression problems
# Can choose: 'identity' to disable any transformation, otherwise use 'auto'
target_transformer = 'auto'

tournament_style = "auto"

# Accuracy equal and above which uses model style if tournament_style = "auto"
#tournament_model_style_accuracy_switch = 6
# Accuracy equal and above which uses feature style if tournament_style = "auto"
# tournament_feature_style_accuracy_switch = 7
#
# Accuracy equal and above which uses fullstack style if tournament_style = "auto"
# tournament_fullstack_style_accuracy_switch = 8
#
# Driverless AI uses a genetic algorithm (GA) to find the best features, best models and
# best hyper parameters for these models. The GA facilitates getting good results while not
# requiring to run/try every possible model/feature/parameter. This version of GA has
# reinforcement learning elements - it uses a form of exploration-exploitation to reach
# optimum solutions. This means it will capitalise on models/features/parameters that seem to
# be working well and continue to exploit them even more, while allowing some room for
# trying new (and semi-random) models/features/parameters to avoid settling on a local
# minimum.
# These models/features/parameters tried are what we call individuals of a population. More
# individuals connote more models/features/parameters to be tried and compete to find the best
# ones.
# num_individuals = 2
#
# set fixed number of individuals (if > 0) - useful to compare different hardware configurations
# fixed_num_individuals = 0
#
# set fixed number of folds (if > 0) when using cross-validation. It may be useful for
# quick runs regardless of the data size
# fixed_num_folds = 0
#
# set fixed number of fold reps (if > 0) - useful for quick runs regardless of data
# fixed_fold_reps = 0
#
# set true to force only first fold for models - useful for quick runs regardless of data
# fixed_only_first_fold_model = false
#
# number of unique targets or folds counts after which switch to faster/simpler non-natural
# sorting and print outs
# sanitize_natural_sort_limit = 1000
#
# Whether target encoding is generally enabled
# Target encoding refers to several different feature transformations (primarily focused on
# categorical data) that aim to represent the feature using information of the actual
# target variable. A simple example can be to use the mean of the target to replace each
# unique category of a categorical feature. This type of features can be very predictive,
# but are prone to overfitting and require more memory as they need to store mappings of
# the unique categories and the target values.
# enable_target_encoding = true
#
# Driverless AI categorises all data (feature engineering) transformers
# More information for these transformers can be viewed here:
# http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/transformations.html
# This section allows excluding/blocking these transformations and may be useful when
# simpler (more interpretable) models are sought at the expense of accuracy.
# the interpretability setting)
# for multi-class: "["NumCatTETransformer", "TextLinModelTransformer",
# "FrequentTransformer", "CVTargetEncodeF", "ClusterDistTransformer",
# "WeightOfEvidenceTransformer", "TruncSVDNumTransformer", "CVCatNumEncodeF",
# "DatesTransformer", "TextTransformer", "FilterTransformer",
# "NumToCatWoETransformer", "NumToCatTETransformer", "ClusterTETransformer",
# "BulkInteractionsTransformer"]"
# for regression/binary: "["TextTransformer", "ClusterDistTransformer",
# "FilterTransformer", "TextLinModelTransformer", "NumToCatTETransformer",
# "DatesTransformer", "WeightOfEvidenceTransformer", "BulkInteractionsTransformer",
# "FrequentTransformer", "CVTargetEncodeF", "NumCatTETransformer",
# "NumToCatWoETransformer", "TruncSVDNumTransformer", "ClusterTETransformer",
# "CVCatNumEncodeF"]"
This list appears in the experiment logs (search for "Transformers used")
# e.g. to disable all Target Encoding: exclude_transformers =
# "['NumCatTETransformer', 'CVTargetEncodeDef', 'NumToCatTETransformer',
# 'ClusterTETransformer']"
#exclude_transformers = ""

Exclude list of genes (i.e. genes (built on top of transformers) to not use,
# independent of the interpretability setting)
Some transformers are used by multiple genes, so this allows different control over feature_
→engineering

for multi-class: "['BulkInteractionsGene', 'WeightOfEvidenceGene',
# 'NumToCatTargetEncodeSingleGene', 'FilterGene', 'TextGene', 'FrequentGene',
# 'NumToCatWeightOfEvidenceGene', 'NumToCatWeightOfEvidenceMonotonicGene',
# CvTargetEncodeSingleGene', 'DateGene', 'NumToCatTargetEncodeMultiGene', '
# DateTimeGene', 'TextLinRegressorGene', 'ClusterIDTargetEncodeSingleGene',
# 'CVCatNumEncodeGene', 'TruncSvdNumGene', 'ClusterIDTargetEncodeMultiGene',
# 'NumCatTargetEncodeMultiGene', 'CVTargetEncodeMultiGene', 'TextLinClassifierGene',
# 'NumCatTargetEncodeSingleGene', 'ClusterDistGene']"

for regression/binary: "['CVTargetEncodeSingleGene', 'NumToCatTargetEncodeSingleGene',
# 'CVCatNumEncodeGene', 'ClusterIDTargetEncodeSingleGene', 'TextLinRegressorGene',
# 'CVTargetEncodeMultiGene', 'ClusterDistGene', 'FilterGene', 'DateGene',
# 'Genus', 'IDTargetEncodeMultiGene', 'NumToCatTargetEncodeMultiGene',
# 'NumCatTargetEncodeMultiGene', 'CVTargetEncodeMultiGene', 'TextLinClassifierGene',
# 'DateTimeGene', 'NumToCatWeightOfEvidenceGene',
# 'NumToCatWeightOfEvidenceMonotonicGene', 'NumCatTargetEncodeSingleGene']"

This list appears in the experiment logs (search for "Genes used")
# e.g. to disable bulk interaction gene, use: exclude_genes =
# "['BulkInteractionsGene']"
#exclude_genes = ""

Parameters for LightGBM to override DAI parameters
parameters should be given as XGBoost equivalent unless unique LightGBM parameter
# e.g. 'eval_metric' instead of 'metric' should be used
# e.g. params_lightgbm = "{'objective': 'binary:logistic', 'n_estimators': 100, 'max_leaves': 64,  
→'random_state': 1234}"  
# e.g. params_lightgbm = { 'n_estimators': 600, 'learning_rate': 0.1, 'reg_alpha': 0.0, 'reg_
→_lamsbs': 0.5, 'gamma': 0, 'max_depth': 0, 'max_bin': 128, 'max_leaves': 256, 'scale_pos_weight
→': 1.0, 'max_delta_step': 3.469919910597877, 'min_child_weight': 1, 'subsample': 0.9,  
→'colsample_bytree': 0.3, 'tree_method': 'gpu_hist', 'grow_policy': 'lossguide', 'min_data_in_
→bin': 3, 'min_child_samples': 5, 'early_stopping_rounds': 20, 'num_classes': 2, 'objective':
→'binary:logistic', 'eval_metric': 'logloss', 'random_state': 987654, 'early_stopping_threshold
→': 0.01, 'monotonicity_constraints': False, 'silent': True, 'debug_verbose': 0, 'subsample_freq
→': 1}"
# avoid including "system"-level parameters like 'n_gpus': 1, 'gpu_id': 0, , 'n_jobs': 1,  
→'booster': 'lightgbm'
# also likely should avoid parameters like: 'objective': 'binary:logistic', unless one really_
→knows what one is doing (e.g. alternative objectives)
# See: https://xgboost.readthedocs.io/en/latest/parameter.html  
And see: https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst  
# Can also pass objective parameters if choose (or in case automatically chosen) certain
→objectives  
# https://lightgbm.readthedocs.io/en/latest/Parameters.html#metric-parameters  
#params_lightgbm = ""["

Parameters for XGBoost to override DAI parameters
similar parameters as lightgbm since lightgbm parameters are transcribed from xgboost_
→equivalent versions
# e.g. params_xgboost = { 'n_estimators': 100, 'max_leaves': 64, 'max_depth': 0, 'random_state':
→1234}"
# See: https://xgboost.readthedocs.io/en/latest/parameter.html  
#params_xgboost = ""[

# Like params_xgboost but for XGBoost's dart method
#params_dart = "{}"

# Parameters for Tensorflow to override DAI parameters
# e.g. params_tensorflow = "{'lr': 0.01, 'add_wide': False, 'add_attention': True, 'epochs': 30, ...
# layers': [100, 100], 'activation': 'selu', 'batch_size': 64, 'chunk_size': 1000, 'dropout': 0.0, ...
# strategy': 'one_shot', 'l1': 0.0, 'l2': 0.0, 'ort_loss': 0.5, 'ort_loss_tau': 0.01, ...
# normalize_type': 'streaming')"

# See: https://keras.io/ , e.g. for activations: https://keras.io/activations/
# Example layers: [500, 500, 500], [100, 100, 100], [100, 100], [50, 50]
# Strategies: '1cycle' or 'one_shot', See: https://github.com/fastai/fastai
# normalize_type: 'streaming' or 'global' (using sklearn StandardScaler)
#params_tensorflow = "{}"

# Parameters for XGBoost's gblinear to override DAI parameters
# e.g. params_gblinear = "{'n_estimators': 100}"
# See: https://xgboost.readthedocs.io/en/latest/parameter.html
#params_gblinear = "{}"

# Parameters for Rulefit to override DAI parameters
# e.g. params_rulefit = "{'max_leaves': 64}"
# See: https://xgboost.readthedocs.io/en/latest/parameter.html
#params_rulefit = "{}"

# Parameters for FTRL to override DAI parameters
#params_ftrl = "{}"

# Dictionary of key:lists of values to use for LightGBM tuning, overrides DAI's choice per key
# e.g. params_tune_lightgbm = "{'min_child_samples': [1,2,5,100,1000], 'min_data_in_bin': [1,2,3, ...
# 10,100,1000]}"
#params_tune_lightgbm = "{}"

# Like params_tune_lightgbm but for XGBoost
# e.g. params_tune_xgboost = "{'max_leaves': [8, 16, 32, 64]}"
#params_tune_xgboost = "{}"

# Like params_tune_lightgbm but for XGBoost's Dart
# e.g. params_tune_dart = "{'max_leaves': [8, 16, 32, 64]}"
#params_tune_dart = "{}"

# Like params_tune_lightgbm but for TensorFlow
# e.g. params_tune_tensorflow = "{'layers': [[10,10,10], [10, 10, 10, 10]]}"
#params_tune_tensorflow = "{}"

# Like params_tune_lightgbm but for gblinear
# e.g. params_tune_gblinear = "{'reg_lambda': [.01, .001, .0001, .0002]}"
#params_tune_gblinear = "{}"

# Like params_tune_lightgbm but for rulefit
# e.g. params_tune_rulefit = "{'max_depth': [4, 5, 6]}"
#params_tune_rulefit = "{}"

# Like params_tune_lightgbm but for ftrl
#params_tune_ftrl = "{}"

# Whether to force max_leaves and max_depth to be 0 if grow_policy is depthwise and lossguide, respectively.
#params_tune_grow_policy_simple_trees = true

# Whether to enable XGBoost models (auto/on/off)
#enable_xgboost = "auto"

# Internal threshold for number of rows x number of columns to trigger no xgboost models due to high memory use
# Overridden if enable_xgboost = "on", in which case always allow xgboost to be used
#xgboost_threshold_data_size_large = 100000000
# Internal threshold for number of rows x number of columns to trigger no xgboost models due to limits on GPU memory capability
# Overridden if enable_xgboost = "on", in which case always allow xgboost to be used
#xgboost_gpu_threshold_data_size_large = 30000000

# Whether to enable GLM models (auto/on/off)
#enable_glm = "auto"

# Whether to enable LightGBM models (auto/on/off)
#enable_lightgbm = "auto"

# Whether to enable Random Forest (in LightGBM package) models (auto/on/off/only)
#enable_rf = "auto"

# Whether to enable TensorFlow models (beta version, no mojo) (auto/on/off)
#enable_tensorflow = "off"

# Whether to enable RuleFit support (beta version, no mojo) (auto/on/off)
#enable_rulefit = "off"

# Whether to enable FTRL support (beta version, no mojo) (auto/on/off)
#enable_ftrl = "off"

# Maximum number of GBM trees or GLM iterations
# Early-stopping usually chooses less
#max_nestimators = 3000

# factor by which max_nestimators is reduced for tuning and feature evolution
#max_nestimators_feature_evolution_factor = 0.2

# Maximum tree depth (and corresponding max max_leaves as 2**max_max_depth)
#max_max_depth = 12

# Default max_bin for tree methods
#default_max_bin = 256

# Default max_bin for lightgbm (recommended for GPU lightgbm)
#default_lightgbm_max_bin = 64

# Maximum max_bin for any tree
#max_max_bin = 256

# Minimum max_bin for any tree
#min_max_bin = 32

# Amount of memory which can handle max_bin = 256 can handle 125 columns and max_bin = 32 for 1000 columns
# As available memory on system goes higher than this scale, can handle proportionally more columns at higher max_bin
# Currently set to 10GB
#scale_mem_for_max_bin = 10737418240

# Factor by which rf gets more depth than gbdt
#factor_rf = 1.5

# Upper limit on learning rate for GBM models
# If want to override min_learning_rate and min_learning_rate_final, set this to smaller value
#max_learning_rate = 0.5

# Lower limit on learning rate for feature engineering GBM models
#min_learning_rate = 0.05

# Lower limit on learning rate for final ensemble GBM models
#min_learning_rate_final = 0.01
# Max. number of epochs for TensorFlow models
tensorflow_max_epochs = 10

# Whether tensorflow will use all CPU cores, or if it will split among all transformers
tensorflow_use_all_cores = true

# Whether tensorflow will use all CPU cores if reproducible is set, or if it will split among all transformers
tensorflow_use_all_cores_even_if_reproducible_true = false

# Max. number of epochs for TensorFlow models for making NLP features
tensorflow_max_epochs_nlp = 2

# Accuracy setting equal and above which will add all enabled TensorFlow NLP models below at start of experiment for text dominated problems
tensorflow_nlp_accuracy_switch = 5

# Whether to use Word-based CNN TensorFlow models for NLP if tensorflow enabled
tensorflow_textcnn = false

# Whether to use Word-based Bi-GRU TensorFlow models for NLP if tensorflow enabled
tensorflow_textbigru = false

# Whether to use Character-level CNN TensorFlow models for NLP if tensorflow enabled
tensorflow_charcnn = false

# Max number of rules to be used for RuleFit models (-1 for all)
rulefit_max_num_rules = -1

# Max tree depth for RuleFit models
rulefit_max_tree_depth = 6

# Max number of trees for RuleFit models
rulefit_max_num_trees = 100

# Internal threshold for number of rows x number of columns to trigger no rulefit models due to being too slow currently
rulefit_threshold_data_size_large = 100000000

# Enable One-Hot-Encoding (which does binning to limit to number of bins to no more than 100 anyway) for categorical columns with fewer than this many unique values
# Set to 0 to disable
one_hot_encoding_cardinality_threshold = 50

# list of possible bins for target encoding (first is default value)
te_bin_list = [25, 10, 100, 250]

# list of possible bins for weight of evidence encoding (first is default value)
# If only want one value: woe_bin_list = [2]
woe_bin_list = [25, 10, 100, 250]

# list of possible bins for one hot encoding (first is default value)
ohe_bin_list = [10, 25, 50, 75, 100]

# Enable time series recipe
time_series_recipe = true

# earliest datetime for automatic conversion of integers in %Y%m%d format to a time column during parsing
min_ymd_timestamp = 19700101

# lastest datetime for automatic conversion of integers in %Y%m%d format to a time column during parsing
max_ymd_timestamp = 20300101
# maximum number of data samples (randomly selected rows) for date/datetime format detection
#max_rows_datetime_format_detection = 100000

# Whether to enable train/valid and train/test distribution shift detection
#check_distribution_shift = true

# Normalized training variable importance above which to check the feature for shift
# Useful to avoid checking likely unimportant features
#shift_key_features_varimp = 0.01

# Whether to only check certain features based upon the value of shift_key_features_varimp
#check_reduced_features = true

# Number of trees to use to train model to check shift in distribution
# No larger than max_nestimators
#shift_trees = 100

# The value of max_bin to use for trees to use to train model to check shift in distribution
#shift_max_bin = 256

# The value of max_depth to use for trees to use to train model to check shift in distribution
#shift_max_depth = 4

# If distribution shift detection is enabled, show features for which shift AUC is above this value
# (AUC of a binary classifier that predicts whether given feature value belongs to train or test data)
#detect_features_distribution_shift_threshold_auc = 0.55

# If distribution shift detection is enabled, drop features for which shift AUC is above this value
# (AUC of a binary classifier that predicts whether given feature value belongs to train or test data)
#drop_features_distribution_shift_threshold_auc = 0.6

# Minimum number of features to keep, keeping least shifted feature at least if 1
#drop_features_distribution_shift_min_features = 1

# Whether to enable detailed traces (in GUI Trace)
#detailed_traces = false

# How close to the optimal value (usually 1 or 0) does the validation score need to be to be considered perfect (to stop the experiment)?
#abs_tol_for_perfect_score = 1e-4

#############################################################################
##Time Series settings

# Normalized probability of choosing to lag non-targets relative to targets
#prob_lag_non_targets = 0.1

# Unnormalized probability of choosing other lag time-series transformers based on interactions
#prob_lagsinteraction = 0.1

# Unnormalized probability of choosing other lag time-series transformers based on aggregations
#prob_lagsaggregates = 0.1

# Automatically generate is-holiday features from date columns
#holiday_features = true

# County code to use to look up holiday calendar (Python package 'holiday')
#holiday_country = "US"

# Max. sample size for automatic determination of time series train/valid split properties, only if time column is selected
#max_time_series_properties_sample_size = 1000000

(continues on next page)
# Maximum number of lag sizes, which are sampled from if sample_lag_sizes==true, else all are → taken (-1 == automatic)
#max_lag_sizes = -1

# Minimum required autocorrelation threshold for a lag to be considered for feature engineering
#min_lag_autocorrelation = 0.1

# How many samples of lag sizes to use for a single time group (single time series signal)
#max_signal_lag_sizes = 100

# Whether to sample lag sizes
#sample_lag_sizes = false

# Probability for new Lags/EMMA gene to use default lags (determined by frequency/gap/horizon, → independent on data)
#prob_default_lags = 0.2

# How many samples of lag sizes to use, chosen randomly out of original set of lag sizes
#max_sampled_lag_sizes = 10

# Override lags to be used
# e.g. [7, 14, 21] # this exact list
# e.g. 21 # produce from 1 to 21
# e.g. 21:3 produce from 1 to 21 in step of 3
# e.g. 5-21 produce from 5 to 21
# e.g. 5-21:3 produce from 5 to 21 in step of 3
#override_lag_sizes = []

# Whether to consider time groups columns as potential features
#allow_tgc_memorization = false

# Maximum time t spent to generate training holdout predictions
# t < 0: up to 1440 minutes (24h) spent for generating training holdout predictions
# t = 0: no training holdout predictions
# t > 0: up to t minutes spent for generating training holdout predictions
#time_series_holdout_predictions_timebank = 0

# MLI (Machine Learning Interpretability) settings
# When number of rows are above this limit sample for MLI for scoring UI data
#mli_sample_above_for_scoring = 1000000

# When number of rows are above this limit sample for MLI for training surrogate models
#mli_sample_above_for_training = 100000

# When sample for MLI how many rows to sample
#mli_sample_size = 100000

# how many bins to do quantile binning
#mli_num_quantiles = 10

# mli random forest number of trees
#mli_drf_num_trees = 100

# Whether to speed up predictions used inside MLI with a fast approximation
#mli_fast_approx = true

# mli number of trees for fast_approx during predict for Shapley
#fast_approx_num_trees = 50

# whether to do only 1 fold and 1 model of all folds and models if ensemble
#fast_approx_do_one_fold_one_model = true

# mli random forest max depth
#mli_drf_max_depth = 20
Using Driverless AI, Release 1.6.2

# not only sample training, but also sample scoring
mli_sample_training = true

# regularization strength for k-LIME GLM's
klime_lambda = [1e-6, 1e-8]
klime_alpha = 0.0

# mli converts numeric columns to enum when cardinality is <= this value
mli_max_numeric_enum_cardinality = 25

# Maximum number of features allowed for k-LIME k-means clustering
mli_max_number_cluster_vars = 6

# Use all columns for k-LIME k-means clustering (this will override 'mli_max_number_cluster_vars')
use_all_columns_klime_kmeans = false

# Strict version check for MLI
mli_strict_version_check = true

# MLI cloud name
mli_cloud_name = ""

# Machine Learning Output : What kinds of files are written related to the machine learning process

# Whether to dump every scored individual's variable importance (both derived and original) to csv/tabulated/json file
# produces files like: individual_scored_id%d.iter%d*features*
dump_varimp_every_scored_indiv = false

# Whether to dump every scored individual's model parameters to csv/tabulated file
# produces files like: individual_scored_id%d.iter%d*params*
dump_modelparams_every_scored_indiv = false

# Whether to append (false) or have separate files (true) for modelparams every scored indiv
#dump_modelparams_separate_files = false

# Location of the AutoDoc template
autodoc_template = "report_template.docx"

# Whether to compute training, validation, and test correlation matrix (table and heatmap pdf) and save to disk
# alpha: currently single threaded and slow for many columns
compute_correlation = false

# Whether to dump to disk a correlation heatmap
produce_correlation_heatmap = false

# Value to report high correlation between original features
high_correlation_value_to_report = 0.95

# Whether to dump to *timings.txt files timing for each transformer
write_trans_timings = true

# whether to delete preview timings if wrote transformer timings
delete_preview_trans_timings = true

# Whether to delete preview cache on server exit
preview_cache_upon_server_exit = true

6.3. Sample Config.toml File
# Instance Local file system
# The option disable access to DAI data_directory from file browser
file_hide_data_directory = true
# The option specify include only list of absolute path prefixes
# which will be only accessible in file browser.
# For example:
file_path_filter_include = ":['/data','/home/michal/']"
#file_path_filter_include = "[]"

# Enable usage of path filters
file_path_filtering_enabled = false

## HDFS
## Note that if using Kerberos, be sure that the DAI time
## is synched with the Kerberos server.
# Configurations for a HDFS data source
# Path of hdfs core-site.xml
core_site_xml_path is deprecated, please use hdfs_config_path
#core_site_xml_path = ""
# HDFS config folder path, can contain multiple config files
#hdfs_config_path = ""
# Path of the principal key tab file
#key_tab_path = ""

# HDFS connector
# Auth type can be Principal/keytab/keytabPrincipal
# Specify HDFS Auth Type, allowed options are:
# noauth : No authentication needed
# principal : Authenticate with HDFS with a principal user
# keytab : Authenticate with a Key tab (recommended). If running
# DAI as a service, then the Kerberos keytab needs to
# be owned by the DAI user.
# keytabimpersonation : Login with impersonation using a keytab
#hdfs_auth_type = "noauth"

# Kerberos app principal user (recommended)
#hdfs_app_principal_user = ""

# Deprecated: Do not use hdfs_app_login_user, user name is taken from user login
#hdfs_app_login_user = ""

# JVM args for HDFS distributions, provide args seperate by space
# -Djava.security.krb5.conf=<path>/krb5.conf
# -Dsun.security.krb5.debug=true
# -Dlog4j.configuration=file:///<path>log4j.properties
#hdfs_app_jvm_args = ""
# hdfs class path
#hdfs_app_classpath = ""
# Limit files returned from HDFS
#hdfs_max_files_listed = 100

# Blue Data DTap connector settings are similar to HDFS connector settings.
# Specify DTap Auth Type, allowed options are:
# noauth : No authentication needed
# principal : Authenticate with DTap with a principal user
# keytab : Authenticate with a Key tab (recommended). If running
# DAI as a service, then the Kerberos keytab needs to
# be owned by the DAI user.
# keytabimpersonation : Login with impersonation using a keytab
# NOTE: "hdfs_app_classpath" and "core_site_xml_path" are both required to be set for DTap
#tap_auth_type = "noauth"
# Dtap (HDFS) config folder path , can contain multiple config files
#tap_config_path = ""
# Path of the principal key tab file
#tap_key_tab_path = ""
# Kerberos app principal user (recommended)
#tap_app_principal_user = ""
# Specify the user id of the current user here as user@realm
#tap_app_login_user = ""
# JVM args for DTap distributions, provide args seperate by space
#tap_app_jvm_args = ""
# DTap (HDFS) class path. NOTE: set "hdfs_app_classpath" also
#tap_app_classpath = ""

# S3 Connector credentials
#aws_access_key_id = ""
#aws_secret_access_key = ""
#aws_role_arn = ""

# What region to use when none is specified in the s3 url.
# Ignored when aws_s3_endpoint_url is set.
#aws_default_region = ""

# Sets enpoint URL that will be used to access S3.
#aws_s3_endpoint_url = ""

# If set to true S3 Connector will try to to obtain credentials assiciated with
# the role attached to the EC2 instance.
#aws_use_ec2_role_credentials = false

# Starting S3 path displayed in UI S3 browser
#s3_init_path = "s3://h2o-public-test-data/smalldata/"

# GCS Connector credentials
# example (suggested) -- "/licenses/my_service_account_json.json"
#gcs_path_to_service_account_json = ""

# Minio Connector credentials
#minio_endpoint_url = ""
#minio_access_key_id = ""
#minio_secret_access_key = ""

# Snowflake Connector credentials
# Recommended Provide: url, user, password
# Optionally Provide: account, user, password
# Example URL: https://<snowflake_account>.<region>.snowflakecomputing.com
#snowflake_url = ""
#snowflake_user = ""
#snowflake_password = ""
#snowflake_account = ""

# KDB Connector credentials
#kdb_user = ""
#kdb_password = ""
#kdb_hostname = ""
#kdb_port = ""
#kdb_app_classpath = ""
#kdb_app_jvm_args = ""

# Azure Blob Store Connector credentials
#azure_blob_account_name = ""
#azure_blob_account_key = ""
#azure_connection_string = ""

# Notification scripts
# - the variable points to a location of script which is executed at given event in experiment
#-lifecycle

6.3. Sample Config.toml File

(continues on next page)
Using Driverless AI, Release 1.6.2

(continued from previous page)

```toml
# - the script should have executable flag enabled
# - use of absolute path is suggested
# The on experiment start notification script location
#listeners_experiment_start = ""
# The on experiment finished notification script location
#listeners_experiment_done = ""

# Default AWS credentials to be used for scorer deployments.
#deployment_aws_access_key_id = ""
#deployment_aws_secret_access_key = ""
#deployment_aws_bucket_name = ""

# Allow the browser to store e.g. login credentials in login form (set to false for higher
#security)
#allow_form_autocomplete = true

# Enable Projects workspace (alpha version, for evaluation)
#enable_projects = false

#-------------------------------------------
## END
```
Driverless AI provides a number of environment variables that can be passed when starting Driverless AI or specified in a config.toml file. The complete list of variables is in the Using the config.toml File section. The steps for specifying variables vary depending on whether you installed a Driverless AI RPM, DEB, or TAR SH or whether you are running a Docker image.

### 7.1 Setting Variables in Docker Images

Each property must be prepended with DRIVERLESS_AI_. The example below starts Driverless AI with environment variables that enable S3 and HDFS access (without authentication)

```bash
nvidia-docker run \
  --pid=host \
  --init \n  --rm \
  -u `id -u`:`id -g` \
  -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,s3,hdfs" \n  -e DRIVERLESS_AI_AUTHENTICATION_METHOD="local" \n  -e DRIVERLESS_AI_LOCAL_HTPASSWD_FILE="<htpasswd_file_location>" \n  -v /etc/passwd:/etc/passwd:ro \n  -v /etc/group:/etc/group:ro \n  -v `pwd`/data:/data \n  -v `pwd`/log:/log \n  -v `pwd`/license:/license \n  -v `pwd`/tmp:/tmp \n  h2oai/dai-centos7-x86_64:TAG
```

### 7.2 Setting Variables in Native Installs

The config.toml file is available in the etc/dai folder after the RPM, DEB, or TAR SH is installed. Edit the desired variables in this file, and then restart Driverless AI.

The example below shows the environment variables in the config.toml file to set when enabling the S3 and HDFS access (without authentication)

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
```

(continues on next page)
Using Driverless AI, Release 1.6.2

(continued from previous page)

```bash
# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/
config.toml"
```

2. Edit the desired environment variables to enable S3 and HDFS access (without authentication).

```bash
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder
→ path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section
→ below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_
→ account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_
→ json below
# minio : Minio Cloud Storage, remember to configure secret and access key
→ below
# snow : Snowflake Data Warehouse, remember to configure Snowflake
→ credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials
→ below (hostname and port, optionally: username, password, classpath, and
→ jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below
→ (account name, account key)
enabled_file_systems = "file,s3,hdfs"

# authentication_method
# unvalidated : Accepts user id and password, does not validate password
# none : Does not ask for user id or password, authenticated as admin
# pam : Accepts user id and password, Validates user with operating system
# ldap : Accepts user id and password, Validates against an ldap server, look
# local: Accepts a user id and password, Validated against a htpasswd file
→ provided in local_htpasswd_file
# for additional settings under LDAP settings
authentication_method = "local"

# Local password file
# Generating a htpasswd file: see syntax below
# htpasswd -B "<location_to_place_htpasswd_file>" "<username>"
# note: -B forces use of bcrypt, a secure encryption method
local_htpasswd_file = "<htpasswd_file_location>"
```

3. Start Driverless AI. Note that the command used to start Driverless AI varies depending on your install type.

```bash
# Linux RPM or DEB with systemd
sudo systemctl start dai

# Linux RPM or DEB without systemd
sudo -H -u dai /opt/h2oai/dai/run-dai.sh

# Linux TAR SH
./run-dai.sh
```
Chapter Eight

Enabling Data Connectors

Driverless AI provides various data connectors for external data sources. Data sources are exposed in the form of the file systems. Each file system is prefixed by a unique prefix. For example:

- To reference data on S3, use s3://.
- To reference data on HDFS, use the prefix hdfs://.
- To reference data on Azure Blob Store, use https://<storage_name>.blob.core.windows.net.
- To reference data on BlueData Datatap, use dtap://.
- To reference data on Google BigQuery, make sure you know the Google BigQuery dataset and the table that you want to query. Use a standard SQL query to ingest data.
- To reference data on Google Cloud Storage, use gs://
- To reference data on kdb+, use the hostname and the port http://<kdb_server>:<port>
- To reference data on Minio, use http://<endpoint_url>.
- To reference data on Snowflake, use a standard SQL query to ingest data.

Refer to the following sections for more information:

8.1 Using Data Connectors with the Docker Image

Available file systems can be configured via the enabled_file_systems property. Note that each property must be prepended with DRIVERLESS_AI_. Replace TAG below with the image tag.

```bash
nvidia-docker run 
--pid=host 
--init 
--rm 
--shm-size=256m 
-u `id -u`:`id -g` 
-p 12345:12345 
-e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,s3,hdfs,gcs,gbq,kdb,minio,snow,dtap,azrbs" 
-v `pwd`/data:/data 
-v `pwd`/log:/log 
-v `pwd`/license:/license 
-v `pwd`/tmp:/tmp 
h2oai/dai-centos7-ppc64le:TAG
```

The sections that follow shows examples describing how to use environment variables to enable HDFS, S3, Google Cloud Storage, Google Big Query, Minio, Snowflake, kdb+, Azure Blob Store, and BlueData DataTap data sources.
8.1.1 S3 Setup

Driverless AI allows you to explore S3 data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with S3.

Description of Configuration Attributes

- `aws_access_key_id`: The S3 access key ID
- `aws_secret_access_key`: The S3 access key

S3 with No Authentication

This example enables the S3 data connector and disables authentication. It does not pass any S3 access key or secret; however it configures Docker DNS by passing the name and IP of the S3 name node. This allows users to reference data stored in S3 directly using the name node address, for example: `s3://name.node/datasets/iris.csv`. Replace TAG below with the image tag.

```bash
nvidia-docker run 
   --shm-size=256m 
   --add-host name.node:172.16.2.186 
   -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,s3" 
   -p 12345:12345 
   --init -it --rm 
   -v /tmp/dtmp:/tmp 
   -v /tmp/dlog:/log 
   -v /tmp/dlicense:/license 
   -v /tmp/ddata:/data 
   -u $(id -u):$(id -g) 
   h2oai/dai-centos7-x86_64:TAG
```

S3 with Authentication

This example enables the S3 data connector with authentication by passing an S3 access key ID and an access key. It also configures Docker DNS by passing the name and IP of the S3 name node. This allows users to reference data stored in S3 directly using the name node address, for example: `s3://name.node/datasets/iris.csv`. Replace TAG below with the image tag.

```bash
nvidia-docker run 
   --shm-size=256m 
   --add-host name.node:172.16.2.186 
   -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,s3" 
   -e DRIVERLESS_AI_AWS_AUTH="True" 
   -e DRIVERLESS_AI_AWS_ACCESS_KEY_ID="<access_key_id>" 
   -e DRIVERLESS_AI_AWS_SECRET_ACCESS_KEY="<access_key>" 
   -p 12345:12345 
   --init -it --rm 
   -v /tmp/dtmp:/tmp 
   -v /tmp/dlog:/log 
   -v /tmp/dlicense:/license 
   -v /tmp/ddata:/data 
   -u $(id -u):$(id -g) 
   h2oai/dai-centos7-x86_64:TAG
```
8.1.2 HDFS Setup

Driverless AI allows you to explore HDFS data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with HDFS.

Supported Hadoop Platforms

- CDH 5.4
- CDH 5.5
- CDH 5.6
- CDH 5.7
- CDH 5.8
- CDH 5.9
- CDH 5.10
- CDH 5.13
- CDH 5.14
- CDH 6.0
- CDH 6.1
- HDP 2.2
- HDP 2.3
- HDP 2.4
- HDP 2.5
- HDP 2.6

Description of Configuration Attributes

- **hdfs_config_path**: The location the HDFS config folder path. This folder can contain multiple config files.
- **hdfs_auth_type**: Selects HDFS authentication. Available values are:
  - **principal**: Authenticate with HDFS with a principal user.
  - **keytab**: Authenticate with a keytab (recommended). If running DAI as a service, then the Kerberos keytab needs to be owned by the DAI user.
  - **keytabimpersonation**: Login with impersonation using a keytab.
  - **noauth**: No authentication needed.
- **key_tab_path**: The path of the principal key tab file. For use when `hdfs_auth_type=principal`.
- **hdfs_app_principal_user**: The Kerberos application principal user.
- **hdfs_app_jvm_args**: JVM args for HDFS distributions. Separate each argument with spaces.
  - `-Djava.security.krb5.conf`
  - `-Dsun.security.krb5.debug`
Using Driverless AI, Release 1.6.2

- -Dlog4j.configuration
  • hdfs_app_classpath: The HDFS classpath.

HDFS with No Authentication

This example enables the HDFS data connector and disables HDFS authentication. It does not pass any HDFS configuration file; however it configures Docker DNS by passing the name and IP of the HDFS name node. This allows users to reference data stored in HDFS directly using name node address, for example: hdfs://name.node/datasets/iris.csv. Replace TAG below with the image tag.

```
nvidia-docker run \
  --pid=host \
  --init \
  --rm \
  --shm-size=256m \
  --add-host name.node:172.16.2.186 \
  -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,hdfs" \
  -e DRIVERLESS_AI_HDFS_AUTH_TYPE='noauth' \
  -e DRIVERLESS_AI_PROCSY_PORT=8080 \
  -p 12345:12345 \
  -v /etc/passwd:/etc/passwd:ro \
  -v /etc/group:/etc/group:ro \
  -v /tmp/dtmp:/tmp \
  -v /tmp/dlog:/log \
  -v /tmp/dlicense:/license \
  -v /tmp/ddata:/data \
  -u $(id -u):$(id -g) \
  h2oai/dai-centos7-x86_64:TAG
```

HDFS with Keytab-Based Authentication

Notes:

- If using Kerberos Authentication, the time on the Driverless AI server must be in sync with Kerberos server. If the time difference between clients and DCs are 5 minutes or higher, there will be Kerberos failures.

- If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user; otherwise Driverless AI will not be able to read/access the Keytab and will result in a fallback to simple authentication and, hence, fail.

This example:

- Places keytabs in the /tmp/dtmp folder on your machine and provides the file path as described below.
- Configures the environment variable DRIVERLESS_AI_HDFS_APP_PRINCIPAL_USER to reference a user for whom the keytab was created (usually in the form of user@realm).

Replace TAG below with the image tag.

```
# Docker instructions
nvidia-docker run \
  --pid=host \
  --init \
  --rm \
  --shm-size=256m \
  -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,hdfs" \
  (continues on next page)
```
HDFS with Keytab-Based Impersonation

Notes:

- If using Kerberos, be sure that the Driverless AI time is synched with the Kerberos server.
- If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user.
- Logins are case sensitive when keytab-based impersonation is configured.

The example:

- Sets the authentication type to `keytabimpersonation`.
- Places keytabs in the `/tmp/dtmp` folder on your machine and provides the file path as described below.
- Configures the `DRIVERLESS_AI_HDFS_APP_PRINCIPAL_USER` variable, which references a user for whom the keytab was created (usually in the form of `user@realm`).

Replace TAG below with the image tag.

```bash
# Docker instructions
nvidia-docker run \n  --pid=host \n  --init \n  --rm \n  --shm-size=256m \n  -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,hdfs" \n  -e DRIVERLESS_AI_HDFS_AUTH_TYPE='keytabimpersonation' \n  -e DRIVERLESS_AI_KEY_TAB_PATH='/tmp/<<keytabname>>' \n  -e DRIVERLESS_AI_HDFS_APP_PRINCIPAL_USER='<<appuser@kerberosrealm>>' \n  -e DRIVERLESS_AI_PROCSY_PORT=8080 \n  -p 12345:12345 \n  -v /etc/passwd:/etc/passwd:ro \n  -v /etc/group:/etc/group:ro \n  -v /tmp/dtmp:/tmp \n  -v /tmp/dlog:/log \n  -v /tmp/dlicense:/license \n  -v /tmp/ddata:/data \n  -u $(id -u):$(id -g) \n  h2oai/dai-centos7-x86_64:TAG
```
Specifying a Hadoop Platform

The following example shows how to build an H2O-3 Hadoop image and run Driverless AI on that image. This example uses CDH 6.0. Change the `H2O_TARGET` to specify a different platform.

1. Clone and then build H2O-3 for CDH 6.0.

   ```bash
   git clone https://github.com/h2oai/h2o-3.git
cd h2o-3
./gradlew clean build -x test
export H2O_TARGET=cdh6.0
export BUILD_HADOOP=true
./gradlew clean build -x test
   ```

2. Start Driverless AI.

   ```bash
docker run -it --rm \
-v `pwd`:pwd \
-w `pwd` \
--entrypoint bash \
--network=host \
-p 8020:8020 \
docker.h2o.ai/cdh-6-w-hive \
-c 'sudo -E startup.sh && \
source /envs/h2o_env_python3.6/bin/activate && 
hadoop jar h2o-hadoop-3/h2o-cdh6.0-assembly/build/libs/h2odriver.jar -\nlibjars "$\{(cat /opt/hive-jars/hive-libjars)\}$" -n 1 -mapperXmx 2g -baseport, 
54445 -notify h2o_one_node -ea -disown && 
export CLOUD_IP=localhost && 
export CLOUD_PORT=54445 && 
make -f scripts/jenkins/Makefile.jenkins test-hadoop-smoke; \
bash'
   ```

3. Run the Driverless AI HDFS connector.

   ```bash
java -cp h2oai-dai-connectors.jar ai.h2o.dai.connectors.HdfsConnector
   ```

4. Verify the commands for `ls` and `cp`, for example.

   ```json
{"coreSiteXmlPath": "/etc/hadoop/conf", "keyTabPath": "", authType: "noauth", 
"srcPath": "hdfs://localhost/user/jenkins/", "dstPath": "/tmp/xxx", 
"command": "cp", "user": "", "appUser": ""}
   ```

8.1.3 Azure Blob Store Setup

Driverless AI allows you to explore Azure Blob Store data sources from within the Driverless AI application. This section describes how to enable the Azure Blob Store data connector in Docker environments.

Description of Configuration Attributes

`azure_blob_account_name`: The Microsoft Azure Storage account name. This should be the dns prefix created when the account was created (for example, “mystorage”).

`azure_blob_account_key`: Specify the account key that maps to your account name.

`azure_connection_string`: Optionally specify a new connection string. With this option, you can include an override for a host, port, and/or account name. For example,
Azure Blob Store Example

This example enables the Azure Blob Store data connector. This allows users to reference data stored on your Azure storage account using the account name, for example: `https://mystorage.blob.core.windows.net`. Replace TAG below with the image tag.

```
nvidia-docker run \
--pid=host \
--init \
--rm \
--shm-size=256m \
-e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,azrbs" \
-e DRIVERLESS_AI_AZURE_BLOB_ACCOUNT_NAME="mystorage" \
-e DRIVERLESS_AI_AZURE_BLOB_ACCOUNT_KEY="<access_key>" \
-p 12345:12345 \
-v /tmp/dtmp://tmp \
-v /tmp/dlog://log \
-v /tmp/dlicense://license \
-v /tmp/ddata://data \
-u $(id -u):$(id -g) \
h2oai/dai-centos7-x86_64:TAG
```

### 8.1.4 BlueData DataTap Setup

This section provides instructions for configuring Driverless AI to work with BlueData DataTap.

**Description of Configuration Attributes**

- **dtap_auth_type**: Selects DTAP authentication. Available values are:
  - `noauth`: No authentication needed
  - `principal`: Authenticate with DataTap with a principal user
  - `keytab`: Authenticate with a Key tab (recommended). If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user.
  - `keytabimpersonation`: Login with impersonation using a keytab

- **dtap_config_path**: The location of the DTAP (HDFS) config folder path. This folder can contain multiple config files. **Note**: The DTAP config file `core-site.xml` needs to contain DTap FS configuration, for example:

```
<configuration>
    <property>
      <name>fs.dtap.impl</name>
      <value>com.bluedata.hadoop.bdfs.Bdfs</value>
      <description>The FileSystem for BlueData dtap: URIs.</description>
    </property>
  </configuration>
```
Using Driverless AI, Release 1.6.2

- dtap_key_tab_path: The path of the principal key tab file. For use when dtap_auth_type=principal.
- dtap_app_principal_user: The Kerberos app principal user (recommended).
- dtap_app_login_user: The user ID of the current user (for example, user@realm).
- dtap_app_jvm_args: JVM args for DTap distributions. Separate each argument with spaces.
- dtap_app_classpath: The DTap classpath.

DataTap with No Authentication

This example enables the DataTap data connector and disables authentication. It does not pass any configuration file; however it configures Docker DNS by passing the name and IP of the DTap name node. This allows users to reference data stored in DTap directly using the name node address, for example: dtap://name.node/datasets/iris.csv or dtap://name.node/datasets/.

(Note: The trailing slash is currently required for directories.) Replace TAG below with the image tag.

```
nvidia-docker run \
    --pid=host \
    --init \
    --rm \
    --shm-size=256m \
    --add-host name.node:172.16.2.186 \
    -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,dtap" \
    -e DRIVERLESS_AI_DTAP_AUTH_TYPE='noauth' \
    -p 12345:12345 \
    -v /etc/passwd:/etc/passwd \
    -v /tmp/dtmp:/tmp \
    -v /tmp/dlog:/log \
    -v /tmp/dlicense:/license \
    -v /tmp/ddata:/data \
    -u $(id -u):$(id -g) \
    h2oai/dai-centos7-x86_64:TAG
```

DataTap with Keytab-Based Authentication

Notes:
- If using Kerberos Authentication, the time on the Driverless AI server must be in sync with Kerberos server. If the time difference between clients and DCs are 5 minutes or higher, there will be Kerberos failures.
- If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user; otherwise Driverless AI will not be able to read/access the Keytab and will result in a fallback to simple authentication and, hence, fail.

This example:
- Places keytabs in the /tmp/dtmp folder on your machine and provides the file path as described below.
- Configures the environment variable DRIVERLESS_AI_DTAP_APP_PRINCIPAL_USER to reference a user for whom the keytab was created (usually in the form of user@realm).

Replace TAG below with the image tag.
# Docker instructions
```
nvidia-docker run
   --pid=host \n   --init \n   --rm \n   --shm-size=256m \n   -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,dtap" \n   -e DRIVERLESS_AI_DTAP_AUTH_TYPE='keytab' \n   -e DRIVERLESS_AI_DTAP_KEY_TAB_PATH='tmp/<<keytabname>>' \n   -e DRIVERLESS_AI_DTAP_APP_PRINCIPAL_USER='<<user@kerberosrealm>>' \n   -p 12345:12345 \n   -v /etc/passwd:/etc/passwd \n   -v /tmp/dtmp:/tmp \n   -v /tmp/dlog:/log \n   -v /tmp/dlicense:/license \n   -v /tmp/ddata:/data \n   -u $(id -u):$(id -g) \n   h2oai/dai-centos7-x86_64:TAG
```

DataTap with Keytab-Based Impersonation

Notes:
- If using Kerberos, be sure that the Driverless AI time is synched with the Kerberos server.
- If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user.

The example:
- Places keytabs in the /tmp/dtmp folder on your machine and provides the file path as described below.
- Configures the DRIVERLESS_AI_DTAP_APP_PRINCIPAL_USER variable, which references a user for whom the keytab was created (usually in the form of \textcode{user@realm}).
- Configures the DRIVERLESS_AI_DTAP_APP_LOGIN_USER variable, which references a user who is being impersonated (usually in the form of \textcode{user@realm}).

Replace TAG below with the image tag.

```
nvidia-docker run
   --pid=host \n   --init \n   --rm \n   --shm-size=256m \n   -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,dtap" \n   -e DRIVERLESS_AI_DTAP_AUTH_TYPE='Keytab' \n   -e DRIVERLESS_AI_DTAP_KEY_TAB_PATH='tmp/<<keytabname>>' \n   -e DRIVERLESS_AI_DTAP_APP_PRINCIPAL_USER='<<appuser@kerberosrealm>>' \n   -e DRIVERLESS_AI_DTAP_APP_LOGIN_USER='<<thisuser@kerberosrealm>>' \n   -p 12345:12345 \n   -v /etc/passwd:/etc/passwd \n   -v /tmp/dtmp:/tmp \n   -v /tmp/dlog:/log \n   -v /tmp/dlicense:/license \n   -v /tmp/ddata:/data \n   -u $(id -u):$(id -g) \n   h2oai/dai-centos7-x86_64:TAG
```

8.1. Using Data Connectors with the Docker Image
8.1.5 Google BigQuery

Driverless AI allows you to explore Google BigQuery data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with Google BigQuery. This setup requires you to enable authentication. If you enable the GCS and/or GBQ connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

In order to enable the GBQ data connector with authentication, you must:

1. Retrieve a JSON authentication file from GCP.
2. Mount the JSON file to the Docker instance.
3. Specify the path to the /json_auth_file.json in the GCS_PATH_TO_SERVICE_ACCOUNT_JSON environmental variable.

Note: The account JSON includes authentications as provided by the system administrator. You can be provided a JSON file that contains both Google Cloud Storage and Google BigQuery authentications, just one or the other, or none at all.

GBQ with Authentication Example

This example enables the GBQ data connector with authentication by passing the JSON authentication file. This assumes that the JSON file contains Google BigQuery authentications. Replace TAG below with the image tag.

```
nvidia-docker run \
  --pid=host \
  --rm \
  --shm-size=256m \
  -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,gbq" \
  -e DRIVERLESS_AI_GCS_PATH_TO_SERVICE_ACCOUNT_JSON="/service_account_json.json" \
  -u `id -u`:`id -g` \
  -p 12345:12345 \
  -v `pwd`/data:/data \
  -v `pwd`/log:/log \
  -v `pwd`/license:/license \
  -v `pwd`/tmp:/tmp \
  -v `pwd`/service_account_json.json:/service_account_json.json \
  h2oai/dai-centos7-x86_64:TAG
```

After Google BigQuery is enabled, you can add datasets by selecting Google BigQuery from the Add Dataset (or Drag and Drop) drop-down menu.
Specify the following information to add your dataset.

1. **Enter BQ Dataset ID with write access to create temporary table**: Enter a dataset ID in Google BigQuery that this user has read/write access to. BigQuery uses this dataset as the location for the new table generated by the query.

   *Note*: Driverless AI’s connection to GBQ will inherit the top-level directory from the service JSON file. So if a dataset named “my-dataset” is in a top-level directory named “dai-gbq”, then the value for the dataset ID input field would be “my-dataset” and not “dai-gbq:my-dataset”.

2. **Enter Google Storage destination bucket**: Specify the name of Google Cloud Storage destination bucket. Note that the user must have write access to this bucket.

3. **Enter Name for Dataset to be saved as**: Specify a name for the dataset, for example, `my_file`.

4. **Enter BigQuery Query (Use StandardSQL)**: Enter a StandardSQL query that you want BigQuery to execute. For example: `SELECT * FROM <my_dataset>.<my_table>`.

5. When you are finished, select the **Click to Make Query** button to add the dataset.
8.1.6 Google Cloud Storage Setup

Driverless AI allows you to explore Google Cloud Storage data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with Google Cloud Storage. This setup requires you to enable authentication. If you enable GCS or GBP connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

In order to enable the GCS data connector with authentication, you must:

1. Obtain a JSON authentication file from GCP.
2. Mount the JSON file to the Docker instance.
3. Specify the path to the /json_auth_file.json in the GCS_PATH_TO_SERVICE_ACCOUNT_JSON environment variable.

**Note:** The account JSON includes authentications as provided by the system administrator. You can be provided a JSON file that contains both Google Cloud Storage and Google BigQuery authentications, just one or the other, or none at all.

GCS with Authentication

This example enables the GCS data connector with authentication by passing the JSON authentication file. This assumes that the JSON file contains Google Cloud Storage authentications. Replace TAG below with the image tag.

```
nvidia-docker run \
  --pid=host \
  --init \ 
  --rm \ 
  --shm-size=256m \ 
  -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,gcs" \ 
  -e DRIVERLESS_AI_GCS_PATH_TO_SERVICE_ACCOUNT_JSON="/service_account_json.json" \ 
  -u `id -u`:`id -g` \ 
  -p 12345:12345 \ 
  -v `pwd`/data:/data \ 
  -v `pwd`/log:/log \ 
  -v `pwd`/license:/license \ 
  -v `pwd`/tmp:/tmp \ 
  -v `pwd`/service_account_json.json:/service_account_json.json \ 
  h2oai/dai-centos7-x86_64:TAG
```

8.1.7 kdb+ Setup

Driverless AI allows you to explore kdb+ data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with kdb+.

Description of Configuration Attributes

- **kdb_user:** (Optional) User name
- **kdb_password:** (Optional) User’s password
- **kdb_hostname:** IP address or host of the KDB server
• `kdb_port`: Port on which the kdb+ server is listening
• `kdb_app_jvm_args`: (Optional) JVM args for kdb+ distributions (for example, `-Dlog4j.configuration`). Separate each argument with spaces.
• `kdb_app_classpath`: (Optional) The kdb+ classpath (or other if the jar file is stored elsewhere).

**kdb+ with No Authentication**

This example enables the kdb+ connector without authentication. The only required flags are the hostname and the port. Replace TAG below with the image tag.

```bash
nvidia-docker run \
--pid=host \ 
--init \ 
--rm \ 
--shm-size=256m \ 
--add-host name.node:172.16.2.186 \ 
-e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,kdb" \ 
-e DRIVERLESS_AI_KDB_HOSTNAME="<ip_or_host_of_kdb_server>" \ 
-e DRIVERLESS_AI_KDB_PORT="<kdb_server_port>" \ 
-p 12345:12345 \ 
-v /tmp/dtmp:/tmp \ 
-v /tmp/dlog:/log \ 
-v /tmp/dlicense:/license \ 
-v /tmp/ddata:/data \ 
-u $(id -u):$(id -g) \ 
h2oai/dai-centos7-x86_64:TAG
```

**kdb+ with Authentication Example**

This example provides users credentials for accessing a kdb+ server from Driverless AI. Replace TAG below with the image tag.

```bash
# Docker instructions
nvidia-docker run \
--pid=host \ 
--init \ 
--rm \ 
--shm-size=256m \ 
-e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,kdb" \ 
-e DRIVERLESS_AI_KDB_HOSTNAME="<ip_or_host_of_kdb_server>" \ 
-e DRIVERLESS_AI_KDB_PORT="<kdb_server_port>" \ 
-e DRIVERLESS_AI_KDB_USER="<username>" \ 
-e DRIVERLESS_AI_KDB_PASSWORD="<password>" \ 
-p 12345:12345 \ 
-v /tmp/dtmp:/tmp \ 
-v /tmp/dlog:/log \ 
-v /tmp/dlicense:/license \ 
-v /tmp/ddata:/data \ 
-u $(id -u):$(id -g) \ 
h2oai/dai-centos7-x86_64:TAG
```

After the kdb+ connector is enabled, you can add datasets by selecting kdb+ from the Add Dataset (or Drag and Drop) drop-down menu.
Specify the following information to add your dataset.

1. **Enter filepath to save query.** Enter the local file path for storing your dataset. For example, `/home/<user>/myfile.csv`. Note that this can only be a CSV file.

2. **Enter KDB Query:** Enter a kdb+ query that you want to execute. Note that the connector will accept any queries. For example: `select from <mytable> or <mytable> lj <myothertable>`

3. When you are finished, select the **Click to Make Query** button to add the dataset.
8.1.8 Minio Setup

This section provides instructions for configuring Driverless AI to work with Minio. Note that unlike S3, authentication must also be configured when the Minio data connector is specified.

Minio with Authentication

This example enables the Minio data connector with authentication by passing an endpoint URL, access key ID, and an access key. It also configures Docker DNS by passing the name and IP of the name node. This allows users to reference data stored in Minio directly using the endpoint URL, for example: http://<endpoint_url>/<bucket>/datasets/iris.csv. Replace TAG below with the image tag.

```bash
nvidia-docker run \
        --shm-size=256m \
        --add-host name.node:172.16.2.186 \
        -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,minio" \
        -e DRIVERLESS_AI_MINIO_ENDPOINT_URL="<endpoint_url>" \
        -e DRIVERLESS_AI_MINIO_ACCESS_KEY_ID="<access_key_id>" \
        -e DRIVERLESS_AI_MINIO_SECRET_ACCESS_KEY="<access_key>" \
        -p 12345:12345 \
        --init -it --rm \
        -v /tmp/dtmp:/tmp \
        -v /tmp/dlog:/log \
        -v /tmp/dlicense:/license \
        -v /tmp/ddata:/data \
        -u $(id -u):$(id -g) \
        h2oai/dai-centos7-x86_64:TAG
```

8.1.9 Snowflake

Driverless AI allows you to explore Snowflake data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with Snowflake. This setup requires you to enable authentication. If you enable Snowflake connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

Snowflake with Authentication Example

This example enables the Snowflake data connector with authentication by passing the account, user, and password variables. Replace TAG below with the image tag.

```bash
nvidia-docker run \
        --rm \
        --shm-size=256m \
        -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,snow" \
        -e DRIVERLESS_AI_SNOWFLAKE_ACCOUNT = "<account_id>" \
        -e DRIVERLESS_AI_SNOWFLAKE_USER = "<username>" \
        -e DRIVERLESS_AI_SNOWFLAKE_PASSWORD = "<password>"\n        -u `id -u`:`id -g` \
        -p 12345:12345 \
        -v `pwd`/data:/data \
        -v `pwd`/log:/log \
        -v `pwd`/license:/license \
        -v `pwd`/tmp:/tmp \
```

(continues on next page)
After the Snowflake connector is enabled, you can add datasets by selecting **Snowflake** from the **Add Dataset (or Drag and Drop)** drop-down menu.

Specify the following information to add your dataset.

1. **Enter Output Filename**: Specify the name of the file on your local system that you want to add to Driverless AI. Note that this can only be a CSV file (for example, `myfile.csv`).

2. **Enter Database**: Specify the name of the Snowflake database that you are querying.

3. **Enter Warehouse**: Specify the name of the Snowflake warehouse that you are querying.

4. **Enter Schema**: Specify the schema of the dataset that you are querying.

5. **Enter Region**: (Optional) Specify the region of the warehouse that you are querying. This can be found in the Snowflake-provided URL to access your database (as in `<optional-deployment-name>.<region>.<cloud-provider>.snowflakecomputing.com`).

6. **Enter Role**: (Optional) Specify your role as designated within Snowflake. See https://docs.snowflake.net/manuals/user-guide/security-access-control-overview.html for more information.

7. **Enter File Formatting Params**: (Optional) Specify any additional parameters for formatting your datasets. Available parameters are listed in https://docs.snowflake.net/manuals/sql-reference/sql/create-file-format.html#optional-parameters. **(Note:** Use only parameters for `TYPE = CSV`.) For example, if your dataset includes a text column that contains commas, you can specify a different delimiter using `FIELD_DELIMITER='character'`. Separate multiple parameters with spaces only. For example:

   ```
   FIELD_DELIMITER='|' FIELD_OPTIONALLY_ENCLOSED_BY=""
   ```

8. **Enter Snowflake Query**: Specify the Snowflake query that you want to execute.
9. When you are finished, select the **Click to Make Query** button to add the dataset.

### 8.2 Using Data Connectors with Native Installs

The `config.toml` file is available in the `etc/dai` folder after the RPM, DEB, or TAR SH is installed. Before enabling a connector, be sure to export the Driverless AI `config.toml` file or add it to `~/.bashrc`. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/
 "config.toml"
```

The sections that follow shows examples describing how to use environment variables in the `config.toml` file to enable HDFS, S3, Google Cloud Storage, Google Big Query, Minio, Snowflake, kdb+, Azure Blob Store, and BlueData DataTap data sources.

#### 8.2.1 S3 Setup

This section provides instructions for configuring Driverless AI to work with S3.
S3 with No Authentication

This example enables the S3 data connector and disables authentication. It does not pass any S3 access key or secret; however it configure Docker DNS by passing the name and IP of the S3 name node. This allows users to reference data stored in S3 directly using name node address, for example: s3://name.node/datasets/iris.csv.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```toml
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

enabled_file_systems = "file, s3"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

S3 with Authentication

This example enables the S3 data connector with authentication by passing an S3 access key ID and an access key. It also configure Docker DNS by passing the name and IP of the S3 name node. This allows users to reference data stored in S3 directly using name node address, for example: s3://name.node/datasets/iris.csv.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.
3. Save the changes when you are done, then stop/restart Driverless AI.

8.2.2 HDFS Setup

This section provides instructions for configuring Driverless AI to work with HDFS.

Supported Hadoop Platforms

- CDH 5.4
- CDH 5.5
- CDH 5.6
- CDH 5.7
- CDH 5.8
- CDH 5.9
- CDH 5.10
- CDH 5.13
- CDH 5.14
- CDH 6.0
• CDH 6.1
• HDP 2.2
• HDP 2.3
• HDP 2.4
• HDP 2.5
• HDP 2.6

Description of Configuration Attributes

• hdfs_config_path: The location the HDFS config folder path. This folder can contain multiple config files.

• hdfs_auth_type: Selects HDFS authentication. Available values are:
  – principal: Authenticate with HDFS with a principal user.
  – keytab: Authenticate with a keytab (recommended). If running DAI as a service, then the Kerberos
            keytab needs to be owned by the DAI user.
  – keytabimpersonation: Login with impersonation using a keytab.
  – noauth: No authentication needed.

• key_tab_path: The path of the principal key tab file. For use when hdfs_auth_type=principal.

• hdfs_app_principal_user: The Kerberos application principal user.

• hdfs_app_login_user: The user ID of the current user (for example, user@realm).

• hdfs_app_jvm_args: JVM args for HDFS distributions. Separate each argument with spaces.
  – -Djava.security.krb5.conf
  – -Dsun.security.krb5.debug
  – -Dlog4j.configuration

• hdfs_app_classpath: The HDFS classpath.

HDFS with No Authentication

This example enables the HDFS data connector and disables HDFS authentication in the config.toml file. This allows users to reference data stored in HDFS directly using the name node address, for example: hdfs://name.node/ datasets/iris.csv.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/
  →config.toml"
```

2. Edit the following environment variables in the config.toml file. Note that the procsy port, which defaults to 12347, also has to be changed.
# IP address and port of procsy process.
procsy_ip = "127.0.0.1"
procsy_port = 8080

# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder
# path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section
# below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account
# json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key
# below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

enabled_file_systems = "file, hdfs"

3. Save the changes when you are done, then stop/restart Driverless AI.

### HDFS with Keytab-Based Authentication

This example:

- Places keytabs in the `/tmp/dtmp` folder on your machine and provides the file path as described below.
- Configures the environment variable `hdfs_app_prinicpal_user` to reference a user for whom the keytab was created (usually in the form of `user@realm`).

1. Export the Driverless AI config.toml file or add it to `.bashrc`. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```bash
# IP address and port of procsy process.
procsy_ip = "127.0.0.1"
procsy_port = 8080

# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder
# path and keytab below
```

(continues on next page)
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account->json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account->json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

```toml
# enabled_file_systems = "file, hdfs"
# HDFS connector
# Auth type can be Principal/keytab/keytabPrincipal
# Specify HDFS Auth Type, allowed options are:
#   noauth : No authentication needed
#   principal : Authenticate with HDFS with a principal user
#   keytab : Authenticate with a Key tab (recommended)
#   keytabimpersonation : Login with impersonation using a keytab

hdfs_auth_type = "keytab"

# Path of the principal key tab file
key_tab_path = "/tmp/<keytabname>"

# Kerberos app principal user (recommended)
hdfs_app_principal_user = "<user@kerberosrealm>"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

### HDFS with Keytab-Based Impersonation

**Notes:**

- If using Kerberos, be sure that the Driverless AI time is synched with the Kerberos server.
- If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user.
- Logins are case sensitive when keytab-based impersonation is configured.

The example:

- Sets the authentication type to `keytabimpersonation`.
- Places keytabs in the `/tmp/dtmp` folder on your machine and provides the file path as described below.
- Configures the `hdfs_app_principal_user` variable, which references a user for whom the keytab was created (usually in the form of `user@realm`).

1. Export the Driverless AI config.toml file or add it to `~/.bashrc`. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
```

(continues on next page)
# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"

2. Edit the following environment variables in the config.toml file.

```toml
# IP address and port of procsy process.
procsy_ip = "127.0.0.1"
procsy_port = 8080

# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder
# path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account
# account.json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account
# account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake
# credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials
# below (hostname and port, optionally: username, password, classpath, and
# jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below
# (account name, account key)
enabled_file_systems = "file, hdfs"

# HDFS connector
# Auth type can be Principal/keytab/keytabPrincipal
# Specify HDFS Auth Type, allowed options are:
# noauth : No authentication needed
# principal : Authenticate with HDFS with a principal user
# keytab : Authenticate with a Key tab (recommended)
# keytabimpersonation : Login with impersonation using a keytab
hdfs_auth_type = "keytabimpersonation"

# Path of the principal key tab file
key_tab_path = "/tmp/<keytabname>"

# Kerberos app principal user (recommended)
hdfs_app_principal_user = "<user@kerberosrealm>"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

### 8.2.3 Azure Blob Store Setup

Driverless AI allows you to explore Azure Blob Store data sources from within the Driverless AI application. This section describes how to enable the Azure Blob Store data connector in native install environments.
Description of Configuration Attributes

azure_blob_account_name: The Microsoft Azure Storage account name. This should be the dns prefix created when the account was created (for example, “mystorage”).

azure_blob_account_key: Specify the account key that maps to your account name.

azure_connection_string: Optionally specify a new connection string. With this option, you can include an override for a host, port, and/or account name. For example,

```
azure_connection_string = "DefaultEndpointsProtocol=http;AccountName=<account_name>;AccountKey=<account_key>;BlobEndpoint=http://<host>:<port>/""
```

Azure Blob Store Example

This example enables the Azure Blob Store data connector. This allows users to reference data stored on your Azure storage account using the account name, for example: https://mystorage.blob.core.windows.net.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account.json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account.json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)
enabled_file_systems = "file, azrbs"

# Azure Blob Store Connector credentials
azure_blob_account_name = "mystorage"
azure_blob_account_key = "<account_key>"
```

3. Save the changes when you are done, then stop/restart Driverless AI.
8.2.4 BlueData DataTap Setup

This section provides instructions for configuring Driverless AI to work with BlueData DataTap.

Description of Configuration Attributes

- **dtap_auth_type**: Selects DTAP authentication. Available values are:
  - noauth: No authentication needed
  - principal: Authenticate with DataTap with a principal user
  - keytab: Authenticate with a Key tab (recommended). If running Driverless AI as a service, then the Kerberos keytab needs to be owned by the Driverless AI user.
  - keytabimpersonation: Login with impersonation using a keytab

- **dtap_config_path**: The location of the DTAP (HDFS) config folder path. This folder can contain multiple config files. **Note**: The DTAP config file core-site.xml needs to contain DTAP FS configuration, for example:

```xml
<configuration>
  <property>
    <name>fs.dtap.impl</name>
    <value>com.bluedata.hadoop.bdfs.Bdfs</value>
    <description>The FileSystem for BlueData dtap: URIs.</description>
  </property>
</configuration>
```

- **dtap_key_tab_path**: The path of the principal key tab file. For use when dtap_auth_type=principal.

- **dtap_app_principal_user**: The Kerberos app principal user (recommended).

- **dtap_app_login_user**: The user ID of the current user (for example, user@realm).

- **dtap_app_jvm_args**: JVM args for DTap distributions. Separate each argument with spaces.

- **dtap_app_classpath**: The DTap classpath.

DataTap with No Authentication

This example enables the DataTap data connector and disables authentication in the config.toml file. This allows users to reference data stored in DataTap directly using the name node address, for example:

dtap://name.node/datasets/iris.csv or dtap://name.node/datasets/.

**Note**: The trailing slash is currently required for directories.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```sh
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder→path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section→below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account→json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account→json below
# minio : Minio Cloud Storage, remember to configure secret and access key→below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials→below (hostname and port, optionally: username, password, classpath, and→jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below→(account name, account key)

enabled_file_systems = "file, dtap"

3. Save the changes when you are done, then stop/restart Driverless AI.

DataTap with Keytab-Based Authentication

This example:

- Places keytabs in the /tmp/dtmp folder on your machine and provides the file path as described below.
- Configures the environment variable dtap_app_prinicpal_user to reference a user for whom the keytab was created (usually in the form of user@realm).

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder→path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section→below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account→json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account→json below
```

(continues on next page)
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)
enabled_file_systems = "file, dtap"

# Blue Data DTap connector settings are similar to HDFS connector settings.
#
# Specify DTap Auth Type, allowed options are:
# noauth : No authentication needed
# principal : Authenticate with DTap with a principal user
# keytab : Authenticate with a Key tab (recommended). If running DAI as a service, then the Kerberos keytab needs to be owned by the DAI user.
# keytabimpersonation : Login with impersonation using a keytab
dtap_auth_type = "keytab"

dtap_key_tab_path = "/tmp/<keytabname>"

# Kerberos app principal user (recommended)
dtap_app_principal_user = "<user@kerberosrealm>"

3. Save the changes when you are done, then stop/restart Driverless AI.

**DataTap with Keytab-Based Impersonation**

The example:
- Places keytabs in the /tmp/dtmp folder on your machine and provides the file path as described below.
- Configures the dtap_app_principal_user variable, which references a user for whom the keytab was created (usually in the form of user@realm).
- Configures the dtap_app_login_user variable, which references a user who is being impersonated (usually in the form of user@realm).

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder below
```
Using Driverless AI, Release 1.6.2

(continued from previous page)

```plaintext
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

enabled_file_systems = "file, dtap"

# Blue Data DTap connector settings are similar to HDFS connector settings.
#
# Specify DTap Auth Type, allowed options are:
#   noauth : No authentication needed
#   principal : Authenticate with DTap with a principal user
#   keytab : Authenticate with a Key tab (recommended). If running DAI as a service, then the Kerberos keytab needs to be owned by the DAI user.
#   keytabimpersonation : Login with impersonation using a keytab

dtap_auth_type = "keytab"

dtap_key_tab_path = "/tmp/<keytabname>"

dtap_app_principal_user = "<user@kerberosrealm>"

dtap_app_login_user = "<user@realm>"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

### 8.2.5 Google Big Query

Driverless AI allows you to explore Google BigQuery data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with Google BigQuery. This setup requires you to enable authentication. If you enable GCS or GBP connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

In order to enable the GBQ data connector with authentication, you must:

1. Obtain a JSON authentication file from GCP.
2. Mount the JSON file to the Docker instance.
3. Specify the path to the `/json_auth_file.json` in the `gcs_path_to_service_account_json` environmental variable.

**Note:** The account JSON includes authentications as provided by the system administrator. You can be provided a JSON file that contains both Google Cloud Storage and Google BigQuery authentications, just one or the other, or
none at all.

**Google BigQuery with Authentication**

This example enables the GBQ data connector with authentication by passing the JSON authentication file. This assumes that the JSON file contains Google BigQuery authentications.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```toml
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder, path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

enabled_file_systems = "file, gbq"

# GCS Connector credentials
# example (suggested) -- "/licenses/my_service_account_json.json"
gcs_path_to_service_account_json = "service_account_json.json"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

After Google BigQuery is enabled, you can add datasets by selecting **Google Big Query** from the **Add Dataset (or Drag and Drop)** drop-down menu.
Specify the following information to add your dataset.

1. **Enter BQ Dataset ID with write access to create temporary table**: Enter a dataset ID in Google BigQuery that this user has read/write access to. BigQuery uses this dataset as the location for the new table generated by the query.

   **Note**: Driverless AI’s connection to GBQ will inherit the top-level directory from the service JSON file. So if a dataset named “my-dataset” is in a top-level directory named “dai-gbq”, then the value for the dataset ID input field would be “my-dataset” and not “dai-gbq:my-dataset”.

2. **Enter Google Storage destination bucket**: Specify the name of Google Cloud Storage destination bucket. Note that the user must have write access to this bucket.

3. **Enter Name for Dataset to be saved as**: Specify a name for the dataset, for example, `my_file`.

4. **Enter BigQuery Query (Use StandardSQL)**: Enter a StandardSQL query that you want BigQuery to execute. For example: `SELECT * FROM <my_dataset>.<my_table>`.

5. When you are finished, select the **Click to Make Query** button to add the dataset.
8.2.6 Google Cloud Storage Setup

This section provides instructions for configuring Driverless AI to work with Google Cloud Storage. This setup requires you to enable authentication. If you enable GCS or GBP connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

In order to enable the GCS data connector with authentication, you must:

1. Obtain a JSON authentication file from GCP.
2. Mount the JSON file to the Docker instance.
3. Specify the path to the /json_auth_file.json in the gcs_path_to_service_account_json environmental variable.

**Note:** The account JSON includes authentications as provided by the system administrator. You can be provided a JSON file that contains both Google Cloud Storage and Google BigQuery authentications, just one or the other, or none at all.

**GCS with Authentication**

This example enables the GCS data connector with authentication by passing the JSON authentication file. This assumes that the JSON file contains Google Cloud Storage authentications.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

   ```bash
   # DEB and RPM
   export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
   # TAR SH
   export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
   ```

2. Edit the following environment variables in the config.toml file.

   ```toml
   # File System Support
   # upload : standard upload feature
   # file : local file system/server file system
   # hdfs : Hadoop file system, remember to configure the HDFS config folder
   # path and keytab below
   # dtap : Blue Data Tap file system, remember to configure the DTap section
   # below
   # s3 : Amazon S3, optionally configure secret and access key below
   # gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
   # gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
   # minio : Minio Cloud Storage, remember to configure secret and access key below
   # snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
   # kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
   # azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)
   ```

   (continues on next page)
enabled_file_systems = "file, gcs"

# GCS Connector credentials
# example (suggested) -- "/licenses/my_service_account_json.json"
gcs_path_to_service_account_json = "/service_account_json.json"

3. Save the changes when you are done, then stop/restart Driverless AI.

### 8.2.7 kdb+ Setup

Driverless AI allows you to explore kdb+ data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with kdb+.

**kdb+ with No Authentication**

This example enables the kdb+ connector without authentication. The only required flags are the hostname and the port.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```plaintext
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```plaintext
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below (account name, account key)

enabled_file_systems = "file, kdb"

# KDB Connector credentials
kdb_user = ""
```

(continues on next page)
kdb_password = ""
kdb_hostname = <ip_or_host_of_kdb_server>
kdb_port = "<kdb_server_port>"
kdb_app_classpath = ""
kdb_app_jvm_args = ""

3. Save the changes when you are done, then stop/restart Driverless AI.

**kdb+ with Authentication Example**

This example provides users credentials for accessing a kdb+ server from Driverless AI.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```plaintext
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Edit the following environment variables in the config.toml file.

```plaintext
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)

enabled_file_systems = "file, kdb"

# kdb+ Connector credentials
kdb_user = "<username>"
kdb_password = "<password>"
kdb_hostname = <ip_or_host_of_kdb_server>
kdb_port = "<kdb_server_port>"
kdb_app_classpath = ""
kdb_app_jvm_args = ""
```

3. Save the changes when you are done, then stop/restart Driverless AI.

After the kdb+ connector is enabled, you can add datasets by selecting kdb+ from the Add Dataset (or Drag and Drop) drop-down menu.
Specify the following information to add your dataset.

1. **Enter filepath to save query.** Enter the local file path for storing your dataset. For example, `/home/<user>/myfile.csv`. Note that this can only be a CSV file.

2. **Enter KDB Query:** Enter a kdb+ query that you want to execute. Note that the connector will accept any queries. For example: `select from <mytable> or <mytable> lj <myothertable>`

3. When you are finished, select the **Click to Make Query** button to add the dataset.
8.2.8 Minio Setup

This section provides instructions for configuring Driverless AI to work with Minio. Note that unlike S3, authentication must also be configured when the Minio data connector is specified.

Minio with Authentication

This example enables the Minio data connector with authentication by passing an endpoint URL, access key ID, and an access key. It also configures Docker DNS by passing the name and IP of the Minio endpoint. This allows users to reference data stored in Minio directly using the endpoint URL, for example: http://<endpoint_url>/<bucket>/datasets/iris.csv.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

   ```
   # DEB and RPM
   export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
   # TAR SH
   export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
   ```

2. Edit the following environment variables in the config.toml file.

   ```
   # File System Support
   # upload : standard upload feature
   # file : local file system/server file system
   # hdfs : Hadoop file system, remember to configure the HDFS config folder below
   # dtap : Blue Data Tap file system, remember to configure the DTap section below
   # s3 : Amazon S3, optionally configure secret and access key below
   # gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
   # gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
   # minio : Minio Cloud Storage, remember to configure secret and access key below
   # snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
   # kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
   # azrbs : Azure Blob Storage, remember to configure Azure credentials below
   # Minio Connector credentials
   minio_endpoint_url = "<endpoint_url>"
   minio_access_key_id = "<access_key_id>"
   minio_secret_access_key = "<access_key>"
   ```

3. Save the changes when you are done, then stop/restart Driverless AI.
Using Driverless AI, Release 1.6.2

8.2.9 Snowflake

Driverless AI allows you to explore Snowflake data sources from within the Driverless AI application. This section provides instructions for configuring Driverless AI to work with Snowflake. This setup requires you to enable authentication. If you enable Snowflake connectors, those file systems will be available in the UI, but you will not be able to use those connectors without authentication.

Snowflake with Authentication

This example enables the Snowflake data connector with authentication by passing the account, user, and password variables.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/→config.toml"
```

2. Edit the following environment variables in the config.toml file.

```toml
# File System Support
# upload : standard upload feature
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the HDFS config folder, path and keytab below
# dtap : Blue Data Tap file system, remember to configure the DTap section, below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account, account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_account_json below
# minio : Minio Cloud Storage, remember to configure secret and access key below
# snow : Snowflake Data Warehouse, remember to configure Snowflake credentials below (account name, username, password)
# kdb : KDB+ Time Series Database, remember to configure KDB credentials below (hostname and port, optionally: username, password, classpath, and jvm_args)
# azrbs : Azure Blob Storage, remember to configure Azure credentials below

enabled_file_systems = "file, snow"

# Snowlake Connector credentials
snowflake_account = "<account_id>"
snowflake_user = "<username>"
snowflake_password = "<password>"
```

3. Save the changes when you are done, then stop/restart Driverless AI.

After the Snowflake connector is enabled, you can add datasets by selecting Snowflake from the Add Dataset (or Drag and Drop) drop-down menu.
Specify the following information to add your dataset.

1. **Enter Output Filename**: Specify the name of the file on your local system that you want to add to Driverless AI. Note that this can only be a CSV file (for example, `myfile.csv`).

2. **Enter Database**: Specify the name of the Snowflake database that you are querying.

3. **Enter Warehouse**: Specify the name of the Snowflake warehouse that you are querying.

4. **Enter Schema**: Specify the schema of the dataset that you are querying.

5. **Enter Region**: (Optional) Specify the region of the warehouse that you are querying. This can be found in the Snowflake-provided URL to access your database (as in `<optional-deployment-name>.<region>.<cloud-provider>.snowflakecomputing.com`).

6. **Enter Role**: (Optional) Specify your role as designated within Snowflake. See [https://docs.snowflake.net/manuals/user-guide/security-access-control-overview.html](https://docs.snowflake.net/manuals/user-guide/security-access-control-overview.html) for more information.

7. **Enter File Formatting Params**: (Optional) Specify any additional parameters for formatting your datasets. Available parameters are listed in [https://docs.snowflake.net/manuals/sql-reference/sql/create-file-format.html#optional-parameters](https://docs.snowflake.net/manuals/sql-reference/sql/create-file-format.html#optional-parameters). (Note: Use only parameters for TYPE = CSV.) For example, if your dataset includes a text column that contains commas, you can specify a different delimiter using `FIELD_DELIMITER='character'`. Separate multiple parameters with spaces only. For example:

   ```
   FIELD_DELIMITER='|' FIELD_OPTIONALLY_ENCLOSED_BY='"'
   ```

8. **Enter Snowflake Query**: Specify the Snowflake query that you want to execute.

9. When you are finished, select the **Click to Make Query** button to add the dataset.
### Make Snowflake Query:

<table>
<thead>
<tr>
<th>Datasets overview</th>
<th>Enter Snowflake Query (Use SQL):</th>
</tr>
</thead>
<tbody>
<tr>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Enter Output Filename...</td>
<td></td>
</tr>
<tr>
<td>creditcard_test_cat.csv</td>
<td></td>
</tr>
<tr>
<td>creditcard_train_cat.csv</td>
<td></td>
</tr>
<tr>
<td>walmart_train_cat.csv</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter Database...</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter Warehouse...</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter Schema...</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter Region [Optional]</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter Role [Optional]</td>
<td></td>
</tr>
<tr>
<td>--</td>
<td></td>
</tr>
<tr>
<td>Enter File Formatting Params [Optional]</td>
<td></td>
</tr>
</tbody>
</table>
Driverless AI supports LDAP, PAM, Local, none, and unvalidated (default) authentication. These can be configured by specifying the environment variables when starting the Driverless AI Docker image or by setting the appropriate environment variables in the config.toml file.

Note: Driverless AI is also integrated with IBM Spectrum Conductor and supports authentication from Conductor. Contact sales@h2o.ai for more information about using IBM Spectrum Conductor authentication.

9.1 Local Authentication Example

This section describes how to enable local authentication in Driverless AI.

9.1.1 Enabling Local Auth in Docker Images

To enable authentication in Docker images, specify the authentication environment variable that you want to use. Each variable must be prepended with DRIVERLESS_AI_. Replace TAG below with the image tag. The example below starts Driverless AI with environment variables the enable the following:

- Local authentication when starting Driverless AI
- S3 and HDFS access (without authentication)

```bash
nvidia-docker run \
   --pid=host \
   --init \
   --rm \
   --shm-size=256m \
   -p 12345:12345 \
   -u `id -u`:`id -g` \
   -e DRIVERLESS_AI_ENABLED_FILESYSTEMS="file,s3,hdfs" \
   -e DRIVERLESS_AI_AUTHENTICATION_METHOD="local" \
   -e DRIVERLESS_AI_LOCAL_HTTPPASSWD_FILE="<htpasswd_file_location>" \
   -v `pwd`/data:/data \
   -v `pwd`/log:/log \
   -v `pwd`/license:/license \
   -v `pwd`/tmp:/tmp \
   h2oai/dai-centos7-x86_64:TAG
```
9.1.2 Enabling Local Auth in the config.toml File for Native Installs

Native installs include DEBs, RPMs, and TAR SH installs. The example below shows the environment variables in the config.toml file to set when enabling the following:

- Local authentication when starting Driverless AI
- S3 and HDFS access (without authentication)

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

2. Open the config.toml file and edit the authentication variables. The config.toml file is available in the etc/dai folder after the RPM or DEB is installed.

```
# File System Support
# file : local file system/server file system
# hdfs : Hadoop file system, remember to configure the hadoop coresite and keytab below
# s3 : Amazon S3, optionally configure secret and access key below
# gcs : Google Cloud Storage, remember to configure gcs_path_to_service_account_json below
# gbq : Google Big Query, remember to configure gcs_path_to_service_account_json below
enabled_file_systems = "file,s3,hdfs"

# authentication_method
# unvalidated : Accepts user id and password, does not validate password
# none : Does not ask for user id or password, authenticated as admin
# pam : Accepts user id and password, Validates user with operating system
# ldap : Accepts user id and password, Validates against an ldap server, look for additional settings under LDAP settings
# local: Accepts a user id and password, Validated against a htpasswd file provided in local_htpasswd_file
authentication_method = "local"

# Local password file
# Generating a htpasswd file: see syntax below
# htpasswd -B "<location_to_place_htpasswd_file>" "<username>"
# note: -B forces use of bcrypt, a secure encryption method
local_htpasswd_file = "<htpasswd_file_location>"
```

3. Start (or restart) Driverless AI. Note that the command used to start Driverless AI varies depending on your install type.

```
# Linux RPM or DEB with systemd
sudo systemctl start dai

# Linux RPM or DEB without systemd
sudo -H -u dai /opt/h2oai/dai/run-dai.sh

# Linux TAR SH
./run-dai.sh
```
9.2 LDAP Authentication Example

This section describes how to enable Lightweight Directory Access Protocol in Driverless AI. The available parameters can be specified as environment variables when starting the Driverless AI Docker image, or they can be set via the config toml file for native installs. Upon completion, all the users in the configured LDAP should be able to log in to Driverless AI and run experiments, visualize datasets, interpret models, etc.

9.2.1 Description of Configuration Attributes

The following options can be specified when enabling LDAP authentication.

- **ldap_server**: The LDAP server domain or IP
- **ldap_port**: The LDAP server port
- **ldap_bind_dn**: The complete DN of the LDAP bind user
- **ldap_bind_password**: The password for the LDAP bind
- **ldap_tls_file**: The Transport Layer Security (TLS) certificate file location
- **ldap_use_ssl**: Whether to enable (TRUE) or disable (FALSE) SSL
- **ldap_search_base**: The location in the Directory Information Tree (DIT) where the search will start
- **ldap_search_filter**: A string that describes what you are searching for
- **ldap_search_attributes**: LDAP attributes to return from search
- **ldap_user_name_attribute=uid**: Specify the key to find user name

9.2.2 LDAP without SSL

The following examples describe how to enable LDAP without SSL when running Driverless AI in the Docker image or through native installs.

**Setting Environment Variables in Docker Images**

The following example shows how to configure LDAP without SSL when starting the Driverless AI Docker image. Replace TAG below with the image tag.

```
nvidia-docker run \  
   --pid=host \  
   --init \  
   --rm \  
   --shm-size=256m \  
   -p 12345:12345 \  
   -u `id -u`:`id -g` \  
   -e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,s3,hdfs" \  
   -e DRIVERLESS_AI_AUTHENTICATION_METHOD="ldap" \  
   -e DRIVERLESS_AI_LDAP_USE_SSL="false" \  
   -e DRIVERLESS_AI_LDAP_SERVER="ldap.forumsys.com" \  
   -e DRIVERLESS_AI_LDAP_PORT="389" \  
   -e DRIVERLESS_AI_SEARCH_BASE="dc=example,dc=com" \  
   -e DRIVERLESS_AI_LDAP_BASE_DN="cn=read-only-admin,dc=example,dc=com" \  
   -e DRIVERLESS_AI_LDAP_BIND_DN="uid=myuser,ou=users,dc=example,dc=com" \  
   -e DRIVERLESS_AI_LDAP_BIND_PASSWORD=password \  
   -e DRIVERLESS_AI_LDAP_BASE_DN="dc=example,dc=com" \
```
Using Driverless AI, Release 1.6.2

(continued from previous page)

```bash
-e DRIVERLESS_AI_LDAP_SEARCH_FILTER="\(\&\(\text{objectClass=person}\)\(\text{cn=*=}\)\)" \n-e DRIVERLESS_AI_LDAP_USER_NAME_ATTRIBUTE="uid" \n-v `pwd`/data:/data \n-v `pwd`/log:/log \n-v `pwd`/license:/license \n-v `pwd`/tmp:/tmp \nh2oai/dai-centos7-x86_64:TAG
```

Using the config.toml file with Native Installs

The following example shows how to configure LDAP without SSL when starting Driverless AI from a native install. Native installs include DEBs, RPMs, and TAR SH installs.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/ \n → config.toml"
```

2. Enable LDAP authentication without SSL.

```bash
# Enable LDAP authentication
authentication_method = "ldap"
# Specify the LDAP server to connect to
ldap_server = "ldap.forumsys.com"
# Specify the LDAP port to connect to
ldap_port = "389"
# Disable SSL
ldap_use_ssl="false"
# Specify the location in the DIT where the search will start
ldap_search_base = "dc=example,dc=com"
# Specify the LDAP search filter
# This is a string that describes what you are searching for
ldap_search_filter = "\(\&\(\text{objectClass=person}\)\(\text{cn=*=}\)\)"
# Specify the complete DN of the LDAP bind user
ldap_bind_dn = "cn=read-only-admin,dc=example,dc=com"
# Specify the LDAP password for the above user
ldap_bind_dn_password = "password"
# Specify The LDAP prefix to be used for step 1 of the LDAP authentication
# The first step connects to the LDAP server using the user as concatenated
# string of - ldap_user_prefix + ldap_search_user_id + ',' + ldap_ou_dn
ldap_user_name_attribute = "uid"
```

3. Start (or restart) Driverless AI.

Chapter 9. Configuring Authentication
Users can now launch Driverless AI using their LDAP credentials. If authentication is successful, the user can access Driverless AI and run experiments, visualize datasets, interpret models, etc.

9.2.3 LDAP with SSL

These examples show how to enable LDAP authentication with SSL and additional parameters that can be specified as environment variables when starting the Driverless AI Docker image, or they can be set via the config.toml file for native installs. Upon completion, all the users in the configured LDAP should be able to log in to Driverless AI and run experiments, visualize datasets, interpret models, etc.

Setting Environment Variables in Docker Images

Specify the following LDAP environment variables when starting the Driverless AI Docker image. This example enables LDAP authentication and shows how to specify additional options that are used when recipe=1. Replace TAG below with the image tag.

```
nvidia-docker run \
--pid=host \n--init \n--rm \n--shm-size=256m \n-p 12345:12345 \n-u `id -u`:`id -g` \n-e DRIVERLESS_AI_ENABLED_FILE_SYSTEMS="file,s3,hdfs" \n-e DRIVERLESS_AI_AUTHENTICATION_METHOD="ldap" \n-e DRIVERLESS_AI_LDAP_SERVER="ldap.forumsys.com" \n-e DRIVERLESS_AI_LDAP_PORT="389" \n-e DRIVERLESS_AI_LDAP_SEARCH_BASE="dc=example,dc=com" \n-e DRIVERLESS_AI_LDAP_SEARCH_FILTER="\(\&\(objectClass=person\)\(cn=\)*\)" \n-e DRIVERLESS_AI_LDAP_USE_SSL="true" \n-e DRIVERLESS_AI_LDAP_TLS_FILE="/tmp/abc-def-root.cer" \n-e DRIVERLESS_AI_LDAP_BIND_DN="cn=read-only-admin,dc=example,dc=com" \n-e DRIVERLESS_AI_LDAP_BIND_PASSWORD="password" \n-e DRIVERLESS_AI_LDAP_BASE_DN="dc=example,dc=com" \n-e DRIVERLESS_AI_LDAP_USER_NAME_ATTRIBUTE="uid" \n-v `pwd`/data:/data \n-v `pwd`/log:/log \n-v `pwd`/license:/license \n-v `pwd`/tmp:/tmp \nh2oai/dai-centos7-x86_64:TAG
```

Upon successful completion, all the users in the configured LDAP should be able to log in to Driverless AI and run experiments, visualize datasets, interpret models, etc.

Using the config.toml file with Native Installs

Native installs include DEBs, RPMs, and TAR SH installs.

1. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
```

(continues on next page)
2. Enable LDAP authentication with SSL.

```bash
# Enable LDAP authentication
authentication_method = "ldap"

# Specify the LDAP server to connect to
ldap_server = "ldap.forumsys.com"

# Specify the LDAP port to connect to
ldap_port = "389"

# Specify the location in the DIT where the search will start
ldap_search_base = "dc=example,dc=com"

# Specify the LDAP search filter
# This is a string that describes what you are searching for
ldap_search_filter = "(\&\&\(objectClass=person\)\(cn=\*)\)"

# If the LDAP connection to the LDAP server needs an SSL certificate,
# then this needs to be specified
ldap_use_ssl = "True"

# Specify the LDAP TLS file location if SSL is set to True
ldap_tls_file = "/tmp/abc-def-root.cer"

# Complete DN of the LDAP bind user
ldap_bind_dn = "cn=read-only-admin,dc=example,dc=com"

# Specify the LDAP password for the above user
ldap_bind_dn_password = "password"

# Specify a key to find the user name
ldap_user_name_attribute = "uid"
```

3. Start (or restart) Driverless AI. Users can now launch Driverless AI using their LDAP credentials. If authentication is successful, the user can access Driverless AI and run experiments, visualize datasets, interpret models, etc.

### 9.3 PAM Authentication Example

This section describes how to enable Pluggable Authentication Modules (PAM) in Driverless AI.

#### 9.3.1 Enabling PAM in Docker Images

In this example, the host Linux system has PAM enabled for authentication and Docker running on that Linux system. The goal is to enable PAM for Driverless AI authentication while the Linux system hosts the user information.

1. Verify that the username (“eric” in this case) is defined in the Linux system.
Using Driverless AI, Release 1.6.2

2. Start Docker on the Linux Server and enable PAM in Driverless AI. Replace TAG below with the image tag.

   ```bash
   [root@Linux-Server]# docker run \
   --rm \
   -- shm-size=256m \
   -u `id -u`:`id -g` \
   -p 12345:12345 \
   -v `pwd`/config:/config \
   -v `pwd`/data:/data \
   -v `pwd`/log:/log \
   -v `pwd`/license:/license \
   -v `pwd`/tmp:/tmp \
   -v /etc/passwd:/etc/passwd \
   -v /etc/shadow:/etc/shadow \
   -v /etc/pam.d/:/etc/pam.d/ \
   -e DRIVERLESS_AI_AUTHENTICATION_METHOD="pam" \
   h2oai/dai-centos7-x86_64:TAG
   ```

3. Obtain the Driverless AI container ID. This ID is required for the next step and will be different every time Driverless AI is started.

   ```bash
   [root@Linux-Server]# docker ps
   CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES
   8e333475ffd8 opsh2oai/h2oai-runtime "./run.sh" 36 seconds ago Up 35 seconds 192.168.0.1:9090->9090/tcp, 192.168.0.1:12345->12345/tcp, 192.168.0.1:54321->54321/tcp clever_swirles
   ```

4. From the Linux Server, verify that the Docker Driverless AI instance can see the shadow file. The example below references 8e333475ffd8, which is the container ID obtained in the previous step.

   ```bash
   [root@Linux-Server]# docker exec 8e333475ffd8 cat /etc/shadow|grep eric
   eric:$6$inOv3GsQuRanR1H4$kYgys3oc2dQ3u9it02WTvAygiGIgQ/yqOi0s.
   g4F9DM1UJGpruUVoG15G6OD3MrX/3uy4gWf1YJnbJofaAni/::0:99999:7:::
   ```

5. Open a Web browser and navigate to port 12345 on the Linux system that is running the Driverless AI Docker Image. Log in with credentials known to the Linux system. The login information will now be validated using PAM.

### 9.3.2 Enabling PAM in the config.toml File for Native Installs

In this example, the host Linux system has PAM enabled for authentication. The goal is to enable PAM for Driverless AI authentication while the Linux system hosts the user information.

This example shows how to edit the config.toml file to enable PAM. The config.toml file is available in the etc/dai folder after the RPM or DEB is installed. Edit the authentication_method variable in this file to enable PAM authentication, and then restart Driverless AI.

1. Verify that the username (“eric” in this case) is defined in the Linux system.
Using Driverless AI, Release 1.6.2

2. Export the Driverless AI config.toml file or add it to ~/.bashrc. For example:

```bash
# DEB and RPM
export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"

# TAR SH
export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
```

3. Edit the `authentication_method` variable in the config.toml file so that PAM is enabled.

```toml
# authentication_method
# unvalidated : Accepts user id and password, does not validate password
# none : Does not ask for user id or password, authenticated as admin
# pam : Accepts user id and password, Validates user with operating system
# ldap : Accepts user id and password, Validates against an ldap server, look
# local: Accepts a user id and password, Validated against a htpasswd file.
# for additional settings under LDAP settings
authentication_method = "pam"
```

4. Start Driverless AI. Note that the command used to start Driverless AI varies depending on your install type.

```bash
# Linux RPM or DEB with systemd
[root@Linux-Server]# sudo systemctl start dai

# Linux RPM or DEB without systemd
[root@Linux-Server]# sudo -H -u dai /opt/h2oai/dai/run-dai.sh

# Linux TAR SH
[root@Linux-Server]# ./run-dai.sh
```

5. Open a Web browser and navigate to port 12345 on the Linux system that is running Driverless AI. Log in with credentials known to the Linux system (as verified in the first step). The login information will now be validated using PAM.
ENABLED NOTIFICATIONS

Driverless AI can be configured to trigger a user-defined script at the beginning and end of an experiment. This functionality can be used to send notifications to services like Slack or to trigger a machine shutdown.

The `config.toml` file exposes the following variables:

- `listeners_experiment_start`: Registers an absolute location of a script that gets executed at the start of an experiment.
- `listeners_experiment_done`: Registers an absolute location of a script that gets executed when an experiment is finished successfully.

Driverless AI accepts any executable as a script. (For example, a script can be implemented in Bash or Python.) There are only two requirements:

- The specified script can be executed. (i.e., The file has executable flag.)
- The script should be able to accept command line parameters.

### 10.1 Script Interfaces

When Driverless AI executes a script, it passes the following parameters as a script command line:

- Application ID: A unique identifier of a running Driverless AI instance.
- User ID: The identification of the user who is running the experiment.
- Experiment ID: A unique identifier of the experiment.
- Experiment Path: The location of the experiment results.

### 10.2 Example

The following example demonstrates how to use notification scripts to shutdown an EC2 machine that is running Driverless AI after all launched experiments are finished. The example shows how to use a notification script in a Docker container and with native installations. The idea of a notification script is to create a simple counter (i.e., number of files in a directory) that counts the number of running experiments. If counter reaches 0-value, then the specified action is performed.

In this example, we use the AWS command line utility to shut down the actual machine; however, the same functionality can be achieved by executing `sudo poweroff` (if the actual user has password-less sudo capability configured) or `poweroff` (if the script `poweroff` has `setuid` bit set up together with executable bit. For more info, please visit: [https://unix.stackexchange.com/questions/85663/poweroff-or-reboot-as-normal-user.](https://unix.stackexchange.com/questions/85663/poweroff-or-reboot-as-normal-user.)

- **The on_start Script.** This script increases the counter of running experiments.
#!/usr/bin/env bash

app_id="${1}"  
experiment_id="${3}"  
tmp_dir="${TMPDIR:-/tmp}/${app_id}"  
exp_file="${tmp_dir}/${experiment_id}"  

mkdir -p "${tmp_dir}"  
touch "${exp_file}"

• The on_done Script. This script decreases the counter and executes machine shutdown when the counter reaches 0-value.

#!/usr/bin/env bash

app_id="${1}"  
experiment_id="${3}"  
tmp_dir="${TMPDIR:-/tmp}/${app_id}"  
exp_file="${tmp_dir}/${experiment_id}"  

if [ -f "${exp_file}" ]; then  
    rm -f "${exp_file}"  
fi  

running_experiments=$(ls -l "${tmp_dir}" | wc -l)  
if [ "${running_experiments}" -gt 0 ]; then  
    echo "There is still ${running_experiments} running experiments!"  
else  
    echo "No experiments running! Machine is going to shutdown!"  
    # Use instance meta-data API to get instance ID and then use AWS CLI to shutdown the machine  
    # This expects, that AWS CLI is properly configured and has capability to shutdown instances enabled.  
fi

10.2.1 Docker Image Users

1. Copy the config.toml file from inside the Docker image to your local filesystem. (Change nvidia-docker run to docker run for non-GPU environments.)

# In your Driverless AI folder (for example, dai_1.5.1),  
# make config and scripts directories  
mkdir config  
mkdir scripts  

# Copy the config.toml file to the new config directory.  
nvidia-docker run \  
--pid=host \  
--rm \  
--init \  
-u `id -u`:`id -g` \  
-v `pwd`/config:/config 

(continues on next page)
2. Edit the **Notification scripts** section in the `config.toml` file and save your changes. Note that in this example, the scripts are saved to a `dai_VERSION/scripts` folder.

```toml
# Notification scripts
# - the variable points to a location of script which is executed at given event in experiment lifecycle
# - the script should have executable flag enabled
# - use of absolute path is suggested
# The on experiment start notification script location
listeners_experiment_start = "dai_VERSION/scripts/on_start.sh"
# The on experiment finished notification script location
listeners_experiment_done = "dai_VERSION/scripts/on_done.sh"
```

3. Start Driverless AI with the `DRIVERLESS_AI_CONFIG_FILE` environment variable. Make sure this points to the location of the edited config.toml file so that the software finds the configuration file. (Change `nvidia-docker run` to `docker run` for non-GPU environments.)

```bash
nvidia-docker run \
--pid=host \
--init \
--rm \
-u `id -u`:`id -g` \
-e DRIVERLESS_AI_CONFIG_FILE="/config/config.toml" \
-v `pwd`/config:/config \
-v `pwd`/data:/data \
-v `pwd`/log:/log \
-v `pwd`/license:/license \
-v `pwd`/tmp:/tmp \
-v `pwd`/scripts:/scripts \
h2oai/dai-centos7-x86_64:TAG
```

### 10.2.2 Native Install Users

1. Export the Driverless AI `config.toml` file or add it to `~/.bashrc`. For example:

   ```bash
   # DEB and RPM
   export DRIVERLESS_AI_CONFIG_FILE="/etc/dai/config.toml"
   # TAR SH
   export DRIVERLESS_AI_CONFIG_FILE="/path/to/your/unpacked/dai/directory/config.toml"
   ```

2. Edit the **Notification scripts** section in the `config.toml` file to point to the new scripts. Save your changes when you are done.

   ```toml
   # Notification scripts
   # - the variable points to a location of script which is executed at given event in experiment lifecycle
   # - the script should have executable flag enabled
   # - use of absolute path is suggested
   ```
3. Start Driverless AI. Note that the command used to start Driverless AI varies depending on your install type.

```bash
# Deb or RPM with systemd (preferred for Deb and RPM):
# Start Driverless AI.
sudo systemctl start dai

# Deb or RPM without systemd:
# Start Driverless AI.
sudo -H -u dai /opt/h2oai/dai/run-dai.sh

# Tar.sh
# Start Driverless AI
./run-dai.sh
```
LAUNCHING DRIVERLESS AI

1. After Driverless AI is installed and started, open a browser and navigate to <server>:12345.

2. The first time you log in to Driverless AI, you will be prompted to read and accept the Evaluation Agreement. You must accept the terms before continuing. Review the agreement, then click I agree to these terms to continue.

3. Log in by entering unique credentials. For example:
   
   **Username:** h2oai  **Password:** h2oai

   Note that these credentials do not restrict access to Driverless AI; they are used to tie experiments to users. If you log in with different credentials, for example, then you will not see any previously run experiments.

4. As with accepting the Evaluation Agreement, the first time you log in, you will be prompted to enter your License Key. Click the Enter License button, then paste the License Key into the License Key entry field. Click Save to continue. This license key will be saved in the host machine’s /license folder.

   **Note:** Contact sales@h2o.ai for information on how to purchase a Driverless AI license.

Upon successful completion, you will be ready to add datasets and run experiments.
11.1 Resources

The Resources dropdown menu provides you with links to view System Information and the Driverless AI User Guide. From this dropdown menu, you can also download the Python Client and the MOJO2 runtime.

11.2 Messages

A Messages menu option is available in the top menu when you launch Driverless AI. Click this to view news and upcoming events regarding Driverless AI.
The Datasets Overview page is the Driverless AI Home page. This shows all datasets that have been imported. Note that the first time you log in, this list will be empty.

12.1 Supported File Types

Driverless AI supports the following dataset file formats:

- arff
- bin
- bz2
- csv
- dat
- feather
- gz
- jay
- nff
Using Driverless AI, Release 1.6.2

- parquet (See note below)
- tgz
- tsv
- txt
- xls
- xlsx
- xz
- zip

**Note:** For Parquet file formats, if you select to import multiple Parquet files, those files will be imported as multiple datasets. If you select a folder of Parquet files, the folder will be imported as a single dataset. Tools like Spark/Hive export data as multiple Parquet files that are stored in a directory with a user-defined name. For example, if you export with `Spark dataFrame.write.parquet("/data/big_parquet_dataset")`, Spark creates a folder `/data/big_parquet_dataset`, which will contain multiple Parquet files (depending on the number of partitions in the input dataset) + metadata.

### 12.2 Adding Datasets

You can add datasets using one of the following methods:

Drag and drop files from your local machine directly onto this page. Note that this method currently works for files that are less than 10 GB.

or

Click the **Add Dataset (or Drag and Drop)** button to upload or add a dataset.

**Notes:**

- Upload File, File System, HDFS, and S3 are enabled by default. These can be disabled by removing them from the `enabled_file_systems` setting in the `config.toml` file. (Refer to **Using the config.toml File** section for more information.)

- If File System is disabled, the Driverless AI will open local filebrowser by default.

- If Driverless AI was started with data connectors enabled for HDFS, BlueData Datatap, S3, Google Cloud Storage, Google Big Query, Minio, Snowflake, KDB+, and/or Azure Blob Store, then a dropdown will appear allowing you to specify where to begin browsing for the dataset. Refer to **Enabling Data Connectors** for more information.
Notes:

- Datasets must be in delimited text format.
- Driverless AI can detect the following separators: ,;t
- When importing a folder, the entire folder and all of its contents are read into Driverless AI as a single file.
- When importing a folder, all of the files in the folder must have the same columns.
Upon completion, the datasets will appear in the Datasets Overview page. Click on a dataset to open a submenu. From this menu, you can specify to view Details, Split, Visualize, Predict, or Delete a dataset. You can also delete an unused dataset by hovering over it, clicking the X button or Delete option, and then confirming the delete. Note: You cannot delete a dataset that was used in an active experiment. You have to delete the experiment first.

12.3 Dataset Details

To view a summary of a dataset or to preview the dataset, click on the dataset or select the [Click for Actions] button beside the dataset that you want to view, and then click Details from the submenu that appears. This opens the Dataset Details page.

12.3.1 Dataset Details Page

The Dataset Details page provides a summary of the dataset. This summary lists each column that is included in the dataset along with the type (see note below), the count, the mean, minimum, maximum, standard deviation, frequency, and the number of unique values.
**Note:** Driverless AI recognizes the following types: integer, string, real, boolean, and time.

Hover over the top of a column to view a summary of the first 20 rows of that column.

To view information for a specific column, type the column name in the field above the graph.
12.3.2 Dataset Rows Page

To switch the view and preview the dataset, click the Dataset Rows button in the top right portion of the UI. Then click the Dataset Overview button to return to the original view.

12.4 Splitting Datasets

In Driverless AI, you can split a training dataset into test and validation datasets.

Perform the following steps to split a dataset.

1. On the Datasets page, select the [Click for Actions] button beside the dataset that you want to split, and then select Split from the submenu that appears.
2. The Dataset Splitter form displays. Specify an Output Name 1 and an Output Name 2 for the first and second part of the split. (For example, you can name one test and one valid.)

3. Optionally specify a Target column (for stratified sampling), a Fold column (to keep rows belonging to the same group together), and/or a Time column.

4. Use the slider to select a split ratio, or enter a value in the Train/Valid Split Ratio field.

5. Click **Save** when you are done.
Upon completion, the split datasets will be available on the Datasets page.

## 12.5 Visualizing Datasets

Perform one of the following steps to visualize a dataset:

- On the Datasets page, select the [Click for Actions] button beside the dataset that you want to view, and then click Visualize from the submenu that appears.

- Click the Autoviz top menu link to go to the Visualizations list page, click the New Visualization button, then select or import the dataset that you want to visualize.

### 12.5.1 The Visualization Page

The Visualization page shows all available graphs for the selected dataset. Note that the graphs on the Visualization page can vary based on the information in your dataset. You can also view and download logs that were generated during the visualization.
The following is a complete list of available graphs.

- **Correlated Scatterplots**: Correlated scatterplots are 2D plots with large values of the squared Pearson correlation coefficient. In most cases, all possible scatterplots based on pairs of features (variables) are examined for correlations. However, if there are more than 50 numerical columns inside the dataset, Driverless AI randomly selects 50 of them and only examines all pairs of these 50. The displayed plots are ranked according to the correlation. Some of these plots may not look like textbook examples of correlation. The only criterion is that they have a large value of Pearson’s r. (Only variables with Pearson R^2 > 0.95^2 are displayed.) When modeling with these variables, you may want to leave out variables that are perfectly correlated with others.

Note that points in the scatterplot can have different sizes. Because Driverless AI aggregates the data and does not display all points, the bigger the point is, the bigger number of exemplars (aggregated points) the plot covers.

- **Spikey Histograms**: Spikey histograms are histograms with huge spikes. This often indicates an inordinate number of single values (usually zeros) or highly similar values. The measure of “spikeyness” is a bin frequency that is ten times the average frequency of all the bins. You should be careful when modeling (particularly regression models) with spikey variables.

- **Skewed Histograms**: Skewed histograms are ones with especially large skewness (asymmetry). The robust measure of skewness is derived from Groeneveld, R.A. and Meeden, G. (1984), “Measuring Skewness and Kurtosis.” The Statistician, 33, 391-399. Highly skewed variables are often candidates for a transformation (e.g., logging) before use in modeling. The histograms in the output are sorted in descending order of skewness.

- **Varying Boxplots**: Varying boxplots reveal unusual variability in a feature across the categories of a categorical variable. The measure of variability is computed from a robust one-way analysis of variance (ANOVA). Sufficiently diverse variables are flagged in the ANOVA. A boxplot is a graphical display of the fractiles of a distribution. The center of the box denotes the median, the edges of a box denote the lower and upper quartiles, and the ends of the “whiskers” denote that range of values. Sometimes outliers occur, in which case the adjacent whisker is shortened to the next lower or upper value. For variables (features) having only a few values, the boxes can be compressed, sometimes into a single horizontal line at the median.

- **Heteroscedastic Boxplots**: Heteroscedastic boxplots reveal unusual variability in a feature across the categories of a categorical variable. Heteroscedasticity is calculated with a Brown-Forsythe test: Brown, M. B. and Forsythe, A. B. (1974), “Robust tests for equality of variances.” Journal of the American Statistical Association, 69, 364-367. Plots are ranked according to their heteroscedasticity values. A boxplot is a graphical display of the fractiles of a distribution. The center of the box denotes the median, the edges of a box denote the lower and upper quartiles.
upper quartiles, and the ends of the “whiskers” denote that range of values. Sometimes outliers occur, in which case the adjacent whisker is shortened to the next lower or upper value. For variables (features) having only a few values, the boxes can be compressed, sometimes into a single horizontal line at the median.

**Biplots:** A Biplot is an enhanced scatterplot that uses both points and vectors to represent structure simultaneously for rows and columns of a data matrix. Rows are represented as points (scores), and columns are represented as vectors (loadings). The plot is computed from the first two principal components of the correlation matrix of the variables (features). You should look for unusual (non-elliptical) shapes in the points that might reveal outliers or non-normal distributions. And you should look for purple vectors that are well-separated. Overlapping vectors can indicate a high degree of correlation between variables.

**Outliers:** Variables with anomalous or outlying values are displayed as red points in a dot plot. Dot plots are constructed using an algorithm in Wilkinson, L. (1999). “Dot plots.” The American Statistician, 53, 276–281. Not all anomalous points are outliers. Sometimes the algorithm will flag points that lie in an empty region (i.e., they are not near any other points). You should inspect outliers to see if they are miscodings or if they are due to some other mistake. Outliers should ordinarily be eliminated from models only when there is a reasonable explanation for their occurrence.

**Correlation Graph:** The correlation network graph is constructed from all pairwise squared correlations between variables (features). For continuous-continuous variable pairs, the statistic used is the squared Pearson correlation. For continuous-categorical variable pairs, the statistic is based on the squared intraclass correlation (ICC). This statistic is computed from the mean squares from a one-way analysis of variance (ANOVA). The formula is (MSbetween - MSwithin)/(MSbetween + (k - 1)MSwithin), where k is the number of categories in the categorical variable. For categorical-categorical pairs, the statistic is computed from Cramer’s V squared. If the first variable has k1 categories and the second variable has k2 categories, then a k1 x k2 table is created from the joint frequencies of values. From this table, we compute a chi-square statistic. Cramer’s V squared statistic is then (chi-square / n) / min(k1,k2), where n is the total of the joint frequencies in the table. Variables with large values of these respective statistics appear near each other in the network diagram. The color scale used for the connecting edges runs from low (blue) to high (red). Variables connected by short red edges tend to be highly correlated.

**Parallel Coordinates Plot:** A Parallel Coordinates Plot is a graph used for comparing multiple variables. Each variable has its own vertical axis in the plot. Each profile connects the values on the axes for a single observation. If the data contain clusters, these profiles will be colored by their cluster number.

**Radar Plot:** A Radar Plot is a two-dimensional graph that is used for comparing multiple variables. Each variable has its own axis that starts from the center of the graph. The data are standardized on each variable between 0 and 1 so that values can be compared across variables. Each profile, which usually appears in the form of a star, connects the values on the axes for a single observation. Multivariate outliers are represented by red profiles. The Radar Plot is the polar version of the popular Parallel Coordinates plot. The polar layout enables us to represent more variables in a single plot.

**Data Heatmap:** The heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables, and columns represent cases (instances). The data are standardized before display so that small values are yellow and large values are red. The rows and columns are permuted via a singular value decomposition (SVD) of the data matrix so that similar rows and similar columns are near each other.

**Missing Values Heatmap:** The missing values heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables and columns represent cases (instances). The data are coded into the values 0 (missing) and 1 (nonmissing). Missing values are colored red and nonmissing values are left blank (white). The rows and columns are permuted via a singular value decomposition (SVD) of the data matrix so that similar rows and similar columns are near each other.

**Gaps Histogram:** The gaps index is computed using an algorithm of Wainer and Schacht based on work by John Tukey. (Wainer, H. and Schacht, Psychometrika, 43, 2, 203-12.) Histograms with gaps can indicate a mixture of two or more distributions based on possible subgroups not necessarily characterized in the dataset.
You can also view an explanation for each graph by clicking the **Help** button in the lower-left corner of each expanded graph.

The correlation network graph is constructed from all pairwise squared correlations between variables (features). For continuous-continuous variable pairs, the statistic used is the squared Pearson correlation. For continuous-categorical variable pairs, the statistic is based on the squared intraclass correlation (ICC). This statistic is computed from the mean squares from a one-way analysis of variance (ANOVA). The formula is (MSTotal - MSwithin)/(MSbetween + (k - 1)*MSwithin), where k is the number of categories in the categorical variable. For categorical-categorical pairs, the statistic is computed from Cramer’s V squared. If the first variable has k1 categories and the second variable has k2 categories, then a k1 x k2 table is created from the joint frequencies of values. From this table, we compute a chi-square statistic, Cramer’s V squared statistic is then (chi-square / n) / min(k1,k2), where n is the total of the joint frequencies in the table. Variables with large values of these respective statistics appear near each other in the network diagram. The color scale used for the connecting edges runs from low (blue) to high (red). Variables connected by short red edges tend to be highly correlated.
13.1 Before You Begin

This section describes how to run an experiment using the Driverless AI UI. Before you begin, it is best that you understand the available options that you can specify. Note that only a dataset and a target column are required to be specified, but Driverless AI provides a variety of experiment and expert settings that you can use to build your models.

After you have a comfortable working knowledge of these options, proceed to the New Experiments section.

13.1.1 Experiment Settings

This section describes the settings that are available when running an experiment.

Dropped Columns

Dropped columns are columns that you do not want to be used as predictors in the experiment. Note that Driverless AI will automatically drop ID columns and columns that contain a significant number of unique values (above `max_relative_cardinality` in the config.toml file or `Max. allowed fraction of uniques for integer and categorical cols` in Expert settings).

Validation Dataset

The validation dataset is used for tuning the modeling pipeline. If provided, the entire training data will be used for training, and validation of the modeling pipeline is performed with only this validation dataset. This is not generally recommended, but can make sense if the data are non-stationary. In such a case, the validation dataset can help to improve the generalization performance on shifting data distributions.

This dataset must have the same number of columns (and column types) as the training dataset. Also note that if provided, the validation set is not sampled down, so it can lead to large memory usage, even if accuracy=1 (which reduces the train size).

Test Dataset

The test dataset is used for testing the modeling pipeline and creating test predictions. The test set is never used during training of the modeling pipeline. (Results are the same whether a test set is provided or not.) If a test dataset is provided, then test set predictions will be available at the end of the experiment.
Weight Column

Optional: Column that indicates the observation weight (a.k.a. sample or row weight), if applicable. This column must be numeric with values $\geq 0$. Rows with higher weights have higher importance. The weight affects model training through a weighted loss function and affects model scoring through weighted metrics. The weight column is not used when making test set predictions, but a weight column (if specified) is used when computing the test score.

Fold Column

Optional: Column to use to create stratification folds during (cross-)validation, if applicable. Must be of integer or categorical type. Rows with the same value in the fold column represent cohorts, and each cohort is assigned to exactly one fold. This can help to build better models when the data is grouped naturally. If left empty, the data is assumed to be i.i.d. (identically and independently distributed). For example, when viewing data for a pneumonia dataset, `person_id` would be a good Fold Column. This is because the data may include multiple diagnostic snapshots per person, and we want to ensure that the same person’s characteristics show up only in either the training or validation frames, but not in both to avoid data leakage. Note that a fold column cannot be specified if a validation set is used or if a Time Column is specified.

Time Column

Optional: Column that provides a time order (time stamps for observations), if applicable. Can improve model performance and model validation accuracy for problems where the target values are auto-correlated with respect to the ordering (per time-series group).

The values in this column must be a datetime format understood by pandas.to_datetime(), like “2017-11-29 00:30:35” or “2017/11/29”, or integer values. If [AUTO] is selected, all string columns are tested for potential date/datetime content and considered as potential time columns. If a time column is found, feature engineering and model validation will respect the causality of time. If [OFF] is selected, no time order is used for modeling and data may be shuffled randomly (any potential temporal causality will be ignored).

When your data has a date column, then in most cases, specifying [AUTO] for the Time Column will be sufficient. However, if you select a specific date column, then Driverless AI will provide you with an additional side menu. At a minimum, this side menu will allow you to specify the number of weeks you want to predict and, if a test set is provided, after how many weeks do you want to start predicting. These options default to [AUTO]. Or you can select Expert Settings to specify per-group periodicities, such as a time-store group or a time-customer_id group. You can also adjust the period unit, which will in turn change the period size. Available period units include: nanosecond, microsecond, millisecond, second, minute, hour, day, week, month, quarter, year.

Notes:

- Engineered features will be used for MLI when a time series experiment is built. This is because munged time series features are more useful features for MLI compared to raw time series features.
- A Time Column cannot be specified if a Fold Column is specified. This is because both fold and time columns are only used to split training datasets into training/validation, so once you split by time, you cannot also split with the fold column. If a Time Column is specified, then the time group columns play the role of the fold column for time series.
- A Time Column cannot be specified if a validation dataset is used.

Refer to *Time Series in Driverless AI* for more information about time series experiments in Driverless AI and to see a time series example.
Accuracy, Time, and Interpretability Knobs

The experiment preview describes what the Accuracy, Time, and Interpretability settings mean for your specific experiment. This preview will automatically update if any of the knob values change. Following is more detailed information describing how these values affect an experiment.

Accuracy

As accuracy increases (as indicated by the tournament_.* toml settings), Driverless AI gradually adjusts the method for performing the evolution and ensemble. At low accuracy, Driverless AI varies features and models, but they
all compete evenly against each other. At higher accuracy, each independent main model will evolve independently and be part of the final ensemble as an ensemble over different main models. At higher accuracies, Driverless AI will evolve ensemble feature types like Target Encoding on and off that evolve independently. Finally, at highest accuracies, Driverless AI performs both model and feature tracking and ensembles all those variations.

The following table describes how the Accuracy value affects a Driverless AI experiment.

<table>
<thead>
<tr>
<th>Accuracy</th>
<th>Max Rows x Cols</th>
<th>Ensemble Level</th>
<th>Target Transformation</th>
<th>Parameter Tuning Level</th>
<th>Num Individuals</th>
<th>Num Folds</th>
<th>Only First Fold Model</th>
<th>Distribution Check</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100K</td>
<td>0</td>
<td>False</td>
<td>0</td>
<td>Auto</td>
<td>3</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>1M</td>
<td>0</td>
<td>False</td>
<td>0</td>
<td>Auto</td>
<td>3</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>50M</td>
<td>0</td>
<td>True</td>
<td>1</td>
<td>Auto</td>
<td>3</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>100M</td>
<td>0</td>
<td>True</td>
<td>1</td>
<td>Auto</td>
<td>3-4</td>
<td>True</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>200M</td>
<td>1</td>
<td>True</td>
<td>1</td>
<td>Auto</td>
<td>3-4</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>500M</td>
<td>2</td>
<td>True</td>
<td>1</td>
<td>Auto</td>
<td>3-5</td>
<td>True</td>
<td>Yes</td>
</tr>
<tr>
<td>7</td>
<td>750M</td>
<td>&lt;=3</td>
<td>True</td>
<td>2</td>
<td>Auto</td>
<td>3-10</td>
<td>Auto</td>
<td>Yes</td>
</tr>
<tr>
<td>8</td>
<td>1B</td>
<td>&lt;=3</td>
<td>True</td>
<td>3</td>
<td>Auto</td>
<td>4-10</td>
<td>Auto</td>
<td>Yes</td>
</tr>
<tr>
<td>9</td>
<td>2B</td>
<td>&lt;=3</td>
<td>True</td>
<td>3</td>
<td>Auto</td>
<td>4-10</td>
<td>Auto</td>
<td>Yes</td>
</tr>
<tr>
<td>10</td>
<td>10B</td>
<td>&lt;=4</td>
<td>True</td>
<td>3</td>
<td>Auto</td>
<td>4-10</td>
<td>Auto</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Note: A check for a shift in the distribution between train and test is done for accuracy >= 5.

The list below includes more information about the parameters that are used when calculating accuracy.

- **Max Rows x Cols:** The maximum number of rows x columns to use in model training
  - For classification, stratified random row sampling is performed (by target)
  - For regression, random row sampling is performed

- **Ensemble Level:** The level of ensembling done for the final model (if no time column is selected)
  - 0: single model
  - 1: 1x 4-fold models ensembled together
  - 2: 2x 5-fold models ensembled together
  - 3: 5x 5-fold models ensembled together
  - 4: 8x 5-fold models ensembled together
  - If ensemble level > 0, then the final model score shows an error estimate that includes the data generalization error (standard deviation of scores over folds) and the error in the estimate of the score (bootstrap score’s standard deviation with sample size same as data size).
  - For accuracy >= 8, the estimate of the error in the validation score reduces, and the error in the score is dominated by the data generalization error.
  - The estimate of the error in the test score is estimated by the maximum of the bootstrap with sample size equal to the test set size and the validation score’s error.

- **Target Transformation:** Try target transformations and choose the transformation(s) that have the best score(s).
  Possible transformations: identity, unit_box, log, square, square root, double square root, inverse, Anscombe, logit, sigmoid

- **Parameter Tuning Level:** The level of parameter tuning done
  - 0: no parameter tuning
  - 1: 8 different parameter settings
Using Driverless AI, Release 1.6.2

- 2: 16 different parameter settings
- 3: 32 different parameter settings
- 4: 64 different parameter settings
- Optimal model parameters are chosen based on a combination of the model’s accuracy, training speed, and complexity.

- **Num Individuals**: The number of individuals in the population for the genetic algorithms
  - Each individual is a gene. The more genes, the more combinations of features are tried.
  - The number of individuals is automatically determined and can depend on the number of GPUs. Typical values are between 4 and 16.

- **Num Folds**: The number of internal validation splits done for each pipeline
  - If the problem is a classification problem, then stratified folds are created.

- **Only First Fold Model**: Whether to only use the first fold split for internal validation to save time
  - Example: Setting Num Folds to 3 and Only First Fold Model = True means you are splitting the data into 67% training and 33% validation.
  - If “Only First Fold Model” is False, then errors on the score shown during feature engineering include the data generalization error (standard deviation of scores over folds) and the error in the estimate of the score (bootstrap score’s standard deviation with a sample size the same as the data size).
  - If “Only First Fold Model” is True, then errors on the score shown during feature engineering include only the error in the estimate of the score (bootstrap score’s standard deviation with a sample size same as the data size).
  - For accuracy >= 8, the estimate of the error in the score reduces, and the error in the score is dominated by the data generalization error. This provides the most accurate generalization error.

- **Early Stopping Rounds**: Time-based means based upon the Time table below.

- **Distribution Check**: Checks whether validation or test data are drawn from the same distribution as the training data. Note that this is purely informative to the user. Driverless AI does not take information from the test set into consideration during training.

- **Strategy**: Feature selection strategy (to prune-away features that do not clearly give improvement to model score). Feature selection is triggered by interpretability. Strategy = “FS” if interpretability >= 6; otherwise strategy is None.

**Time**

This specifies the relative time for completing the experiment (i.e., higher settings take longer). Early stopping will take place if the experiment doesn’t improve the score for the specified amount of iterations.
Using Driverless AI, Release 1.6.2

<table>
<thead>
<tr>
<th>Time</th>
<th>Iterations</th>
<th>Early Stopping Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1-5</td>
<td>None</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>7</td>
<td>150</td>
<td>15</td>
</tr>
<tr>
<td>8</td>
<td>200</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>300</td>
<td>30</td>
</tr>
<tr>
<td>10</td>
<td>500</td>
<td>50</td>
</tr>
</tbody>
</table>

Note: See the Accuracy table for cases when not based upon time.

Interpretability

In the following tables, **Ensemble Level** is the level of ensembling done for the final model (if no time column is selected).

- 0: single model
- 1: 1x 4-fold models ensembled together
- 2: 2x 5-fold models ensembled together
- 3: 5x 5-fold models ensembled together

If **Monotonicity Constraints** are enabled, the model will satisfy knowledge about monotonicity in the data and monotone relationships between the predictors and the target variable. For example, in house price prediction, the house price should increase with lot size and number of rooms, and should decrease with crime rate in the area. If enabled, Driverless AI will automatically determine if monotonicity is present and enforce it in its modeling pipelines. Depending on the correlation, Driverless AI will assign positive, negative, or no monotonicity constraints. Monotonicity is enforced if the absolute correlation is greater than 0.1. All other predictors will not have monotonicity enforced.

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Ensemble Level</th>
<th>Monotonicity Constraints</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 5</td>
<td>&lt;= 3</td>
<td>Disabled</td>
</tr>
<tr>
<td>&gt;= 6</td>
<td>&lt;= 2</td>
<td>Disabled</td>
</tr>
<tr>
<td>&gt;= 7</td>
<td>&lt;= 2</td>
<td>Enabled</td>
</tr>
<tr>
<td>&gt;= 8</td>
<td>&lt;= 1</td>
<td>Enabled</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>Enabled</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Transformers**</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;= 5</td>
<td>All</td>
</tr>
<tr>
<td>0-5</td>
<td>Interpretability#5 - [TruncSvdNum, ClusterDist]</td>
</tr>
<tr>
<td>0-6</td>
<td>Interpretability#6 - [ClusterTE, ClusterID, IsolationForestAnomaly]</td>
</tr>
<tr>
<td>0-7</td>
<td>Interpretability#7 - [NumToCatTE]</td>
</tr>
<tr>
<td>0-8</td>
<td>Interpretability#8 - [NumCatTE, NumToCatWoE]</td>
</tr>
<tr>
<td>0-9</td>
<td>Interpretability#9 - [BulkInteractions, WeightOfEvidence, CvCatNumEncode, NumToCatWeightOfEvidenceMonotonic]</td>
</tr>
<tr>
<td>0-10</td>
<td>Interpretability#10 - [CVTargetEncodeFit, CVCatNumericEncodeF, Frequent]</td>
</tr>
</tbody>
</table>

** Interpretability# - [lost transformers] explains which transformers are lost by going up by 1 to that interpretability.
** Exception - NumToCatWeightOfEvidenceMonotonic removed for interpretability<=6.

** For interpretability <= 10, i.e. only [Filter for numeric, Frequent for categorical, DateTime for Date+Time, Date for dates, and Text for text]

- **Target Transformers:**
  - For regression, applied on target before any other transformations.

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Target Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=10</td>
<td>TargetTransformer_identity</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>TargetTransformer_unit_box</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>TargetTransformer_sqrt</td>
</tr>
<tr>
<td>&lt;=9</td>
<td>TargetTransformer_log</td>
</tr>
<tr>
<td>&lt;=9</td>
<td>TargetTransformer_square</td>
</tr>
<tr>
<td>&lt;=8</td>
<td>TargetTransformer_double_sqrt</td>
</tr>
<tr>
<td>&lt;=6</td>
<td>TargetTransformer_logit</td>
</tr>
<tr>
<td>&lt;=6</td>
<td>TargetTransformer_sigmoid</td>
</tr>
<tr>
<td>&lt;=5</td>
<td>TargetTransformer_Anscombe</td>
</tr>
<tr>
<td>&lt;=4</td>
<td>TargetTransformer_inverse</td>
</tr>
</tbody>
</table>

- **Date Types Detected:**
  - categorical
  - date
  - datetime
  - numeric
  - text

- **Transformers used on raw features to generate new features:**

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Transformer</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;=10</td>
<td>Filter</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>DateTime</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>Date</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>Text</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>TextLin</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>CvTargetEncodeMulti</td>
</tr>
<tr>
<td>&lt;=10</td>
<td>CvTargetEncodeSingle</td>
</tr>
<tr>
<td>&lt;=9</td>
<td>CvCatNumEncode</td>
</tr>
<tr>
<td>&lt;=9</td>
<td>WeightOfEvidence</td>
</tr>
<tr>
<td>&lt;=9 and &gt;=7</td>
<td>NumToCatWeightOfEvidenceMonotonic</td>
</tr>
<tr>
<td>&lt;=9</td>
<td>BulkInteractions</td>
</tr>
<tr>
<td>&lt;=8</td>
<td>NumToCatWeightOfEvidence</td>
</tr>
<tr>
<td>&lt;=8</td>
<td>NumCatTargetEncodeMulti</td>
</tr>
<tr>
<td>&lt;=8</td>
<td>NumCatTargetEncodeSingle</td>
</tr>
<tr>
<td>&lt;=7</td>
<td>Frequent</td>
</tr>
<tr>
<td>&lt;=7</td>
<td>NumToCatTargetEncodeMulti</td>
</tr>
<tr>
<td>&lt;=7</td>
<td>NumToCatTargetEncodeSingle</td>
</tr>
<tr>
<td>&lt;=6</td>
<td>ClusterIDTargetEncodeMulti</td>
</tr>
<tr>
<td>&lt;=6</td>
<td>ClusterIDTargetEncodeSingle</td>
</tr>
<tr>
<td>&lt;=5</td>
<td>TruncSvdNum</td>
</tr>
<tr>
<td>&lt;=5</td>
<td>ClusterDist</td>
</tr>
</tbody>
</table>
** Default N-way interactions are up to 8-way except:
  
  – BulkInteractions are always 2-way.
  
  – Interactions are minimal-way (e.g. 1-way for CvTargetEncode) if interpretability=10.

• Feature importance threshold below which features are removed

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Threshold</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>varimp_threshold_at_interpretability_10</td>
</tr>
<tr>
<td>9</td>
<td>config.toml varimp_threshold_at_interpretability_10/5.0</td>
</tr>
<tr>
<td>8</td>
<td>varimp_threshold_at_interpretability_10/7.0</td>
</tr>
<tr>
<td>7</td>
<td>varimp_threshold_at_interpretability_10/10.0</td>
</tr>
<tr>
<td>6</td>
<td>varimp_threshold_at_interpretability_10/20.0</td>
</tr>
<tr>
<td>5</td>
<td>varimp_threshold_at_interpretability_10/30.0</td>
</tr>
<tr>
<td>4</td>
<td>varimp_threshold_at_interpretability_10/50.0</td>
</tr>
<tr>
<td>3</td>
<td>varimp_threshold_at_interpretability_10/500.0</td>
</tr>
<tr>
<td>2</td>
<td>varimp_threshold_at_interpretability_10/5000.0</td>
</tr>
<tr>
<td>1</td>
<td>1E-30</td>
</tr>
</tbody>
</table>

** Also used for strategy=FS dropping of features, but the threshold is the above value multiplied by config.varimp_fspermute_factor.

• Base model used for scoring features and building final model

<table>
<thead>
<tr>
<th>Interpretability</th>
<th>Allowed Base Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Only GLM if glm_enable_more==True or glm_enable_exclusive=True, GBM+GLM if glm_enable==True, else only GBM</td>
</tr>
<tr>
<td>9</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>8</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>7</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>6</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>5</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>4</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>3</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>2</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
<tr>
<td>1</td>
<td>GBM unless glm_enable_exclusive=True, GBM+GLM if glm_enable_more==True</td>
</tr>
</tbody>
</table>

** When mixing GBM and GLM in parameter tuning, the search space is split 50%/50% between GBM and GLM.

** Classification, Reproducible, and Enable GPUs Buttons

• **Classification** or **Regression** button. Driverless AI automatically determines the problem type based on the response column. Though not recommended, you can override this setting by clicking this button.

• **Reproducible**: This button allows you to build an experiment with a random seed and get reproducible results. If this is disabled (default), then results will vary between runs.

• **Enable GPUs**: Specify whether to enable GPUs. (Note that this option is ignored on CPU-only systems.)
13.1.2 Expert Settings

This section describes the Expert Settings options that are available when starting an experiment. Note that the default values for these options are derived from the environment variables in the config.toml file. Refer to the Sample Config.toml File section for more information about each of these options.

Note that by default the feature brain pulls in any better model regardless of the features even if the new model disabled those features. For full control over features pulled in via changes in these Expert Settings, users should set the Feature Brain Level option to 0.

Pipeline Building Recipe

Specify the Pipeline Building recipe type. **Auto** (default) specifies that all models and features are automatically determined by experiment settings, config.toml settings, and the feature engineering effort. **Compliant** is similar to **Auto** except for the following:

- Interpretability is forced to be 10.
- Only use GLM or RuleFit.
- Does not convert any numerics to categoricals except via one-hot encoding.
- Doesn’t use any ensemble.
- No feature brain is used.
- Interaction depth is set to 1.
- Target transformer is forced to be identity for regression.
• Doesn’t use distribution shift between train, valid, and test to drop features.

**XGBoost GBM Models**

This option allows you to specify whether to build XGBoost models as part of the experiment (for both the feature engineering part and the final model). XGBoost is a type of gradient boosting method that has been widely successful in recent years due to its good regularization techniques and high accuracy.

**LightGBM Models**

This option allows you to specify whether to build LightGBM models as part of the experiment. LightGBM Models are the default models.

**LightGBM Random Forest Models**

Select **auto**, **on**, **off**, or **only** from this dropdown to specify whether to include LightGBM Random Forest models as part of the experiment.

**GLM Models**

This option allows you to specify whether to build GLM models (generalized linear models) as part of the experiment (usually only for the final model unless it’s used exclusively). GLMs are very interpretable models with one coefficient per feature, an intercept term and a link function.

**TensorFlow Models**

This option allows you to specify whether to build TensorFlow models as part of the experiment (usually only for text features engineering and for the final model unless it’s used exclusively). Enable this option for NLP experiments. TensorFlow models are not yet supported by MOJOs (only Python scoring pipelines are supported).

**RuleFit Models**

This option allows you to specify whether to build RuleFit models as part of the experiment. Note that MOJOs are not yet supported (only Python scoring pipelines). Note that multiclass classification is not yet supported for RuleFit models. Rules are stored to text files in the experiment directory for now.

**FTRL Models**

This option allows you to specify whether to build Follow the Regularized Leader (FTRL) models as part of the experiment. Note that MOJOs are not yet supported (only Python scoring pipelines). FTRL supports binomial and multinomial classification for categorical targets, as well as regression for continuous targets.

**Number of Models During Tuning Phase**

Specify the number of models to tune during pre-evolution phase. Specify a lower value to avoid excessive tuning, or specify a higher to perform enhanced tuning. This option defaults to -1 (auto).
**Ensemble Level**

Specify one of the following ensemble levels:

- **-1** = auto, based upon ensemble_accuracy_switch, accuracy, size of data, etc. (Default)
- **0** = No ensemble, only final single model on validated iteration/tree count. Note that predicted probabilities will not be available. (Refer to the following FAQ.)
- **1** = 1 model, multiple ensemble folds (cross-validation)
- **2** = 2 models, multiple ensemble folds (cross-validation)
- **3** = 3 models, multiple ensemble folds (cross-validation)
- **4** = 4 models, multiple ensemble folds (cross-validation)

**Data Distribution Shift Detection**

Specify whether Driverless AI should detect data distribution shifts between train/valid/test datasets (if provided). Currently, this information is only presented to the user and not acted upon.

**Max Allowed Feature Shift (AUC) Before Dropping Feature**

Specify the maximum allowed AUC value for a feature before dropping the feature.

When train and test differ (or train/valid or valid/test) in terms of distribution of data, then there can be a model built that tells you for each row whether the row is in train or test. That model includes an AUC value. If the AUC is above this specified threshold, then Driverless AI will consider it a strong enough shift to drop features that are shifted.

This option defaults to 0.6.

**Select Target Transformation of the Target for Regression Problems**

Specify whether to automatically select target transformation for regression problems. Selecting **Identity** disables any transformation. This value defaults to **Auto**.

**Enable Target Encoding**

Specify whether to use Target Encoding when building the model. Target encoding is the process of replacing a categorical value with the mean of the target variable. This is enabled by default.

**Time Series Lag-Based Recipe**

This recipe specifies whether to include Time Series lag features when training a model with a provided (or autodetected) time column. Lag features are the primary automatically generated time series features and represent a variable’s past values. At a given sample with time stamp \( t \), features at some time difference \( T \) (lag) in the past are considered. For example if the sales today are 300, and sales of yesterday are 250, then the lag of one day for sales is 250. Lags can be created on any feature as well as on the target. Lagging variables are important in time series because knowing what happened in different time periods in the past can greatly facilitate predictions for the future. More information about time series lag is available in the Time Series Use Case: Sales Forecasting section.
Time Series Lags Override

Specify a lag override value such as 7, 14, 21, etc.

Probability to Create Non-Target Lag Features

Lags can be created on any feature as well as on the target. Specify a probability value for creating non-target lag features.

Make Python Scoring Pipeline

Specify whether to automatically build a Python Scoring Pipeline for the experiment. If enabled, then when the experiment is completed, the Python Scoring Pipeline can be immediately downloaded. If disabled, the Python Scoring Pipeline will have to be built separately after the experiment is complete.

Make MOJO Scoring Pipeline

Specify whether to automatically build a MOJO (Java) Scoring Pipeline for the experiment. If enabled, then when the experiment is completed, the MOJO Scoring Pipeline can be immediately downloaded. If disabled, the MOJO Scoring Pipeline will have to be built separately after the experiment is complete.

Max Number of Rules for RuleFit

Specify the maximum number of rules to be used for RuleFit models. This defaults to -1, which specifies to use all rules.

Feature Brain Level

H2O.ai Brain enables caching and smart re-use (checkpointing) of prior models to generate features for new models. Use this option to specify the brain caching level.

- -1: Don’t use any brain cache.
- 0: Don’t use any brain cache but still write to cache.
• 1: Smart checkpoint if an old experiment_id is passed in (for example, via running “resume one like this” in the GUI).
• 2: Smart checkpoint if the experiment matches all column names, column types, classes, class labels, and time series options identically. (default)
• 3: Smart checkpoint like level #1, but for the entire population. Tune only if the brain population is of insufficient size.
• 4: Smart checkpoint like level #2, but for the entire population. Tune only if the brain population is of insufficient size.
• 5: Smart checkpoint like level #4, but will scan over the entire brain cache of populations (starting from resumed experiment if chosen) in order to get the best scored individuals.

When enabled, the directory where the H2O.ai Brain meta model files are stored is H2O.ai_brain. In addition, the default maximum brain size is 20GB. Both the directory and the maximum size can be changed in the config.toml file.

Quantile-Based Imbalanced Sampling (binary)

Specify whether to enable the quantile-based sampling method for imbalanced binary classification. (Note: Only applicable if the class ratio is above the imbalanced ratio undersampling threshold.) When enabled, model on data is used to create deciles of predictions, and then each decile is sampled from uniformly.

The idea behind quantile-based imbalanced sampling is that we do not want to just randomly down sample the majority class; instead we want to get an interesting representation of the majority class.

Here are the steps used to perform quantile-based imbalanced sampling:

1. Train an initial model.
2. Use the model from Step 1 to score each record in the majority class.
3. Bin the majority class records based on their prediction.
4. Randomly sample records from each bin.

If our use case was fraud, then quantile-based imbalanced sampling would sample not-fraud records based on the prediction of an initial model. This ensures that we have an even distribution of records that are easy to classify as not-fraud (low prediction bins) and records that are harder to classify as not-fraud (high prediction bins).

Generate Holiday Features

For time-series experiments, enable this option to generate holiday features for the experiment.

Random Seed

Specify a random seed for the experiment. When a seed is defined and the reproducible button is enabled (not by default), the algorithm will behave deterministically.

Force 64-bit Precision

Specify whether to enable 64-bit data and transformer precision. This option is disabled by default, as Driverless AI uses 32-bit floats.

This setting shouldn’t matter unless your data value exceeds 1E38 or if you need to resolve more than 1 part in 10 million. If your data has larger numbers than the above, then you may encounter a Best score is not finite: %f % best_score
AssertionError: Best score is not finite: inf
error. This
Using Driverless AI, Release 1.6.2

can be resolved by using data_precision="float64" and transformer_precision="float64" in config.toml or by enabling this option.

Min Number of Rows Needed to Run an Experiment

Specify the minimum number of rows that a dataset must contain in order to run an experiment. This value defaults to 100.

Max Number of Original Features Used

Specify the maximum number of features you want to be selected in an experiment. This defaults to 10000.

Max Number of Engineered Features

Specify the maximum number of features to include in the final model’s feature engineering pipeline. If -1 is specified (default), then Driverless AI will automatically determine the number of features.

Max Number of Rows times the Number of Columns for Feature Evolution Data Splits

Specify the maximum number of rows allowed for feature evolution data splits (not for the final pipeline). This value defaults to 100,000,000.

Feature Engineering Effort (0..10)

Specify a value from 0 to 10 for the Driverless AI feature engineering effort. Higher values generally lead to more time (and memory) spent in feature engineering. This value defaults to 5.

- 0: Keep only numeric features. Only model tuning during evolution.
- 1: Keep only numeric features and frequency-encoded categoricals. Only model tuning during evolution.
- 2: Similar to 1 but instead just no Text features. Some feature tuning before evolution.
- 3: Similar to 5 but only tuning during evolution. Mixed tuning of features and model parameters.
- 4: Similar to 5, but slightly more focused on model tuning.
- 5: Balanced feature-model tuning. (Default)
- 6-7: Similar to 5 but slightly more focused on feature engineering.
- 8: Similar to 6-7 but even more focused on feature engineering with high feature generation rate and no feature dropping even if high interpretability.
- 9-10: Similar to 8 but no model tuning during feature evolution.

Max Feature Interaction Depth

Specify the maximum number of features to be used for interaction features like grouping for target encoding, weight of evidence and other likelihood estimates.

Exploring feature interactions can be important in gaining better predictive performance. The interaction can take multiple forms (i.e. feature1 + feature2 or feature1 * feature2 + ... featureN). Although certain machine learning algorithms (like tree-based methods) can do well in capturing these interactions as part of their training process, still generating them may help them (or other algorithms) yield better performance.
The depth of the interaction level (as in “up to” how many features may be combined at once to create one single feature) can be specified to control the complexity of the feature engineering process. Higher values might be able to make more predictive models at the expense of time. This value defaults to 8.

**Max Allowed Fraction of Uniques for Integer and Categorical Cols**

Specify the maximum fraction of unique values for integer and categorical columns. If the column has a larger fraction of unique values than that, it will be considered an ID column and ignored. This value defaults to 0.95.

**Threshold for String Columns to be Treated as Text**

Specify the threshold value (from 0 to 1) for string columns to be treated as text (0.0 - text; 1.0 - string). This value defaults to 0.3.

**Max TensorFlow Epochs**

When building TensorFlow models, specify the maximum number of epochs to train models with (it might stop earlier). This value defaults to 100. This option is ignored if TensorFlow models is disabled.

**Enable Word-Based CNN TensorFlow Models for NLP**

Specify whether to use Word-based CNN TensorFlow models for NLP. This option is ignored if TensorFlow is disabled.

**Enable Word-Based BiGRU TensorFlow Models for NLP**

Specify whether to use Word-based BiGRU TensorFlow models for NLP. This option is ignored if TensorFlow is disabled.

**Enable Character-Based CNN TensorFlow Models for NLP**

Specify whether to use Character-level CNN TensorFlow models for NLP. This option is ignored if TensorFlow is disabled.

**Max TensorFlow Epochs for NLP**

When building TensorFlow NLP features (for text data), specify the maximum number of epochs to train feature engineering models with (it might stop earlier). This value defaults to 2. This option is ignored if TensorFlow models is disabled.

**Min DAI Iterations**

Specify the minimum number of Driverless AI iterations for an experiment. This can be used during restarting, when you want to continue for longer despite a score not improving. This defaults to 0.
Max Number of Trees/Iterations

Specify the upper limit on the number of trees (GBM) or iterations (GLM) for all tree models. This defaults to 3000. Depending on accuracy settings, a fraction of this limit will be used.

Reduction Factor for Number of Trees/Iterations During Feature Evolution

Specify the factor by which max_nestimators is reduced for tuning and feature evolution. This option defaults to 0.2. So by default, Driverless AI will produce no more than 0.2 * 3000 trees/iterations during feature evolution.

Max Learning Rate for Tree Models

Specify the maximum learning rate for tree models during feature engineering. Larger values can speed up feature engineering, but can hurt accuracy. This value defaults to 0.5.

Number of Cores to Use

Specify the number of cores to use for the experiment. Note that if you specify -1, then all available cores will be used. Lower values can reduce memory usage, but might slow down the experiment.

#GPUs/Model

Specify the number of GPUs to user per model, with -1 meaning all GPUs per model. In all cases, XGBoost tree and linear models use the number of GPUs specified per model, while LightGBM and Tensorflow revert to using 1 GPU/model and run multiple models on multiple GPUs.

Note: FTRL does not use GPUs. Rulefit uses GPUs for parts involving obtaining the tree using LightGBM.

#GPUs/Experiment

Specify the number of GPUs to use per experiment. A value of -1 specifies to use all available GPUs. Must be at least as large as the number of GPUs to use per model (or -1).

GPU Starting ID

Specify the GPU starting ID (0 is the first GPU).

Compute Correlation Matrix

Specify whether to compute training, validation, and test correlation matrix (table and heatmap pdf) and save to disk. This option is disabled by default.

Threshold for Reporting High Correlation

Specify the value to report high correlation between original features. This defaults to 0.95.
Enable Detailed Scored Model Info

Specify whether to dump every scored individual’s model parameters to a csv/tabulated file. If enabled, Driverless AI produces files such as “individual_scored_id%d.iter%d*params*”. This option is disabled by default.

Enable Detailed Scored Features Info

Specify whether to dump every scored individual’s variable importance (both derived and original) to a csv/tabulated/json file. If enabled, Driverless AI produces files such as “individual_scored_id%d.iter%d*features*”. This option is disabled by default.

Enable Detailed Traces

Specify whether to enable detailed tracing in Driverless AI trace when running an experiment. This is disabled by default.

Add to config.toml via toml String

Specify any additional configuration overrides from the config.toml file that you want to include in the experiment. (Refer to the Sample Config.toml File section to view options that can be overridden during an experiment.) Setting this will override all other settings. Separate multiple config overrides with \n. For example, the following enables Poisson distribution for LightGBM and disables Target Transformer Tuning. Note that in this example double quotes are escaped (" ").

```
params_lightgbm="{"objective":"poisson"}"
\n\ntarget_transformer=identity
```

Or you can specify config overrides similar to the following without having to escape double quotes:

```
"enable_glm="off" \nenable_xgboost="off" \nenable_lightgbm="off" \nenable_tensorflow="on"
""\nmax_cores=10 \ndata_precision="float32" \nmax_rows_feature_evolution=5000000000 \nensemble_accuracy_switch=11 \nfeature_engineering_effort=1 \n\ntarget_transformer="identity" \ntournament_feature_style_accuracy_switch=5 \nparams_tensorflow="{"layers": [100, 100, 100, 100, 100, 100]}"
```

When running the Python client, config overrides would be set as follows:

```
model = h2o.start_experiment_sync(
    dataset_key=train.key,
    target_col='target',
    is_classification=True,
    accuracy=7,
    time=5,
    interpretability=1,
    config_overrides="",
    feature_brain_level=0
    enable_lightgbm="off"
    enable_xgboost="off"
    enable_ftrl="off"
"
)
```
13.1.3 Scorers

Classification or Regression

- **GINI** (Gini Coefficient): The Gini index is a well-established method to quantify the inequality among values of a frequency distribution, and can be used to measure the quality of a binary classifier. A Gini index of zero expresses perfect equality (or a totally useless classifier), while a Gini index of one expresses maximal inequality (or a perfect classifier).

  The Gini index is based on the Lorenz curve. The Lorenz curve plots the true positive rate (y-axis) as a function of percentiles of the population (x-axis).

  The Lorenz curve represents a collective of models represented by the classifier. The location on the curve is given by the probability threshold of a particular model. (i.e., Lower probability thresholds for classification typically lead to more true positives, but also to more false positives.)

  The Gini index itself is independent of the model and only depends on the Lorenz curve determined by the distribution of the scores (or probabilities) obtained from the classifier.
Regression

- **R2 (R Squared)**: The R2 value represents the degree that the predicted value and the actual value move in unison. The R2 value varies between 0 and 1 where 0 represents no correlation between the predicted and actual value and 1 represents complete correlation.

Calculating the R2 value for linear models is mathematically equivalent to $1 - \frac{SSE}{SST}$ (or $1 - \text{residual sum of squares/total sum of squares}$). For all other models, this equivalence does not hold, so the $1 - \frac{SSE}{SST}$ formula cannot be used. In some cases, this formula can produce negative R2 values, which is mathematically impossible for a real number. Because Driverless AI does not necessarily use linear models, the R2 value is calculated using the squared Pearson correlation coefficient.

R2 equation:
Using Driverless AI, Release 1.6.2

\[ R^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2 \sum_{i=1}^{n}(y_i - \bar{y})^2}} \]

Where:
- \( x \) is the predicted target value
- \( y \) is the actual target value

**MSE (Mean Squared Error):** The MSE metric measures the average of the squares of the errors or deviations. MSE takes the distances from the points to the regression line (these distances are the “errors”) and squaring them to remove any negative signs. MSE incorporates both the variance and the bias of the predictor.

MSE also gives more weight to larger differences. The bigger the error, the more it is penalized. For example, if your correct answers are 2, 3, 4 and the algorithm guesses 1, 4, 3, then the absolute error on each one is exactly 1, so squared error is also 1, and the MSE is 1. But if the algorithm guesses 2, 3, 6, then the errors are 0, 0, 2, the squared errors are 0, 0, 4, and the MSE is a higher 1.333. The smaller the MSE, the better the model’s performance. (Tip: MSE is sensitive to outliers. If you want a more robust metric, try mean absolute error (MAE).)

MSE equation:

\[ MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2 \]

**RMSE (Root Mean Squared Error):** The RMSE metric evaluates how well a model can predict a continuous value. The RMSE units are the same as the predicted target, which is useful for understanding if the size of the error is of concern or not. The smaller the RMSE, the better the model’s performance. (Tip: RMSE is sensitive to outliers. If you want a more robust metric, try mean absolute error (MAE).)

RMSE equation:

\[ RMSE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2} \]

Where:
- \( N \) is the total number of rows (observations) of your corresponding dataframe.
- \( y \) is the actual target value.
- \( \hat{y} \) is the predicted target value.

**RMSLE (Root Mean Squared Logarithmic Error):** This metric measures the ratio between actual values and predicted values and takes the log of the predictions and actual values. Use this instead of RMSE if an under-prediction is worse than an over-prediction. You can also use this when you don’t want to penalize large differences when both of the values are large numbers.

RMSLE equation:

\[ RMSLE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (ln\left(\frac{y_i + 1}{\hat{y}_i + 1}\right))^2} \]

Where:
- \( N \) is the total number of rows (observations) of your corresponding dataframe.
• \( y \) is the actual target value.
• \( \hat{y} \) is the predicted target value.

- **RMSPE** (Root Mean Square Percentage Error): This metric is the RMSE expressed as a percentage. The smaller the RMSPE, the better the model performance.

  RMSPE equation:

  \[
  RMSPE = \sqrt{\frac{1}{N} \sum_{i=1}^{N} \frac{(y_i - \hat{y}_i)^2}{(y_i)^2}}
  \]

- **MAE** (Mean Absolute Error): The mean absolute error is an average of the absolute errors. The MAE units are the same as the predicted target, which is useful for understanding whether the size of the error is of concern or not. The smaller the MAE the better the model’s performance. (Tip: MAE is robust to outliers. If you want a metric that is sensitive to outliers, try root mean squared error (RMSE).)

  MAE equation:

  \[
  MAE = \frac{1}{N} \sum_{i=1}^{N} |x_i - \hat{x}_i|
  \]

  Where:
  - \( N \) is the total number of errors
  - \( |x_i - x| \) equals the absolute errors.

- **MAPE** (Mean Absolute Percentage Error): MAPE measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

  MAPE equation:

  \[
  MAPE = \left( \frac{1}{N} \sum_{i=1}^{N} \frac{|Actual - Forecast|}{|Actual|} \right) * 100
  \]

  Because the MAPE measure is in percentage terms, it gives an indication of how large the error is across different scales. Consider the following example:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
<th>Absolute Error</th>
<th>Absolute Percentage Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
<td>80%</td>
</tr>
<tr>
<td>15,000</td>
<td>15,004</td>
<td>4</td>
<td>0.03%</td>
</tr>
</tbody>
</table>

Both records have an absolute error of 4, but this error could be considered “small” or “big” when you compare it to the actual value.

- **SMAPE** (Symmetric Mean Absolute Percentage Error): Unlike the MAPE, which divides the absolute errors by the absolute actual values, the SMAPE divides by the mean of the absolute actual and the absolute predicted values. This is important when the actual values can be 0 or near 0. Actual values near 0 cause the MAPE value to become infinitely high. Because SMAPE includes both the actual and the predicted values, the SMAPE value can never be greater than 200%.

  Consider the following example:

<table>
<thead>
<tr>
<th>Actual</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>0.05</td>
</tr>
<tr>
<td>0.03</td>
<td>0.04</td>
</tr>
</tbody>
</table>
The MAPE for this data is 216.67% but the SMAPE is only 80.95%.

Both records have an absolute error of 4, but this error could be considered “small” or “big” when you compare it to the actual value.

- **MER (Median Error Rate or Median Absolute Percentage Error):** MER measures the median size of the error in percentage terms. It is calculated as the median of the unsigned percentage error.

  MER equation:

  \[
  MER = (\text{median} \left| \frac{\text{Actual} - \text{Forecast}}{\text{Actual}} \right|) \times 100
  \]

  Because the MER is the median, half the scored population has a lower absolute percentage error than the MER, and half the population has a larger absolute percentage error than the MER.

**Classification**

- **MCC (Matthews Correlation Coefficient):** The goal of the MCC metric is to represent the confusion matrix of a model as a single number. The MCC metric combines the true positives, false positives, true negatives, and false negatives using the equation described below.

  A Driverless AI model will return probabilities, not predicted classes. To convert probabilities to predicted classes, a threshold needs to be defined. Driverless AI iterates over possible thresholds to calculate a confusion matrix for each threshold. It does this to find the maximum MCC value. Driverless AI’s goal is to continue increasing this maximum MCC.

  Unlike metrics like Accuracy, MCC is a good scorer to use when the target variable is imbalanced. In the case of imbalanced data, high Accuracy can be found by simply predicting the majority class. Metrics like Accuracy and F1 can be misleading, especially in the case of imbalanced data, because they do not consider the relative size of the four confusion matrix categories. MCC, on the other hand, takes the proportion of each class into account. The MCC value ranges from -1 to 1 where -1 indicates a classifier that predicts the opposite class from the actual value, 0 means the classifier does no better than random guessing, and 1 indicates a perfect classifier.

  MCC equation:

  \[
  MCC = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}
  \]

- **F05, F1, and F2:** A Driverless AI model will return probabilities, not predicted classes. To convert probabilities to predicted classes, a threshold needs to be defined. Driverless AI iterates over possible thresholds to calculate a confusion matrix for each threshold. It does this to find the maximum some F metric value. Driverless AI’s goal is to continue increasing this maximum F metric.

  The F1 score provides a measure for how well a binary classifier can classify positive cases (given a threshold value). The F1 score is calculated from the harmonic mean of the precision and recall. An F1 score of 1 means both precision and recall are perfect and the model correctly identified all the positive cases and didn’t mark a negative case as a positive case. If either precision or recall are very low it will be reflected with a F1 score closer to 0.

  F1 equation:

  \[
  F1 = 2 \left( \frac{(\text{precision}) (\text{recall})}{\text{precision} + \text{recall}} \right)^
  \]
Where:

- **precision** is the positive observations (true positives) the model correctly identified from all the observations it labeled as positive (the true positives + the false positives).

- **recall** is the positive observations (true positives) the model correctly identified from all the actual positive cases (the true positives + the false negatives).

The **F0.5** score is the weighted harmonic mean of the precision and recall (given a threshold value). Unlike the F1 score, which gives equal weight to precision and recall, the F0.5 score gives more weight to precision than to recall. More weight should be given to precision for cases where False Positives are considered worse than False Negatives. For example, if your use case is to predict which products you will run out of, you may consider False Positives worse than False Negatives. In this case, you want your predictions to be very precise and only capture the products that will definitely run out. If you predict a product will need to be restocked when it actually doesn’t, you incur cost by having purchased more inventory than you actually need.

F0.5 equation:

\[
F_{0.5} = 1.25 \left( \frac{(\text{precision}) \cdot (\text{recall})}{0.25 \cdot \text{precision} + \text{recall}} \right)
\]

Where:

- **precision** is the positive observations (true positives) the model correctly identified from all the observations it labeled as positive (the true positives + the false positives).

- **recall** is the positive observations (true positives) the model correctly identified from all the actual positive cases (the true positives + the false negatives).

The **F2** score is the weighted harmonic mean of the precision and recall (given a threshold value). Unlike the F1 score, which gives equal weight to precision and recall, the F2 score gives more weight to recall than to precision. More weight should be given to recall for cases where False Negatives are considered worse than False Positives. For example, if your use case is to predict which customers will churn, you may consider False Negatives worse than False Positives. In this case, you want your predictions to capture all of the customers that will churn. Some of these customers may not be at risk for churning, but the extra attention they receive is not harmful. More importantly, no customers actually at risk of churning have been missed.

F2 equation:

\[
F_2 = 5 \left( \frac{(\text{precision}) \cdot (\text{recall})}{4 \cdot \text{precision} + \text{recall}} \right)
\]

Where:

- **precision** is the positive observations (true positives) the model correctly identified from all the observations it labeled as positive (the true positives + the false positives).

- **recall** is the positive observations (true positives) the model correctly identified from all the actual positive cases (the true positives + the false negatives).

**Accuracy**: In binary classification, Accuracy is the number of correct predictions made as a ratio of all predictions made. In multiclass classification, the set of labels predicted for a sample must exactly match the corresponding set of labels in y_true.

A Driverless AI model will return probabilities, not predicted classes. To convert probabilities to predicted classes, a threshold needs to be defined. Driverless AI iterates over possible thresholds to calculate a confusion matrix for each threshold. It does this to find the maximum Accuracy value. Driverless AI’s goal is to continue increasing this maximum Accuracy.
Accuracy equation:

\[
Accuracy = \left( \frac{\text{number correctly predicted}}{\text{number of observations}} \right)
\]

- **Logloss**: The logarithmic loss metric can be used to evaluate the performance of a binomial or multinomial classifier. Unlike AUC which looks at how well a model can classify a binary target, logloss evaluates how close a model’s predicted values (uncalibrated probability estimates) are to the actual target value. For example, does a model tend to assign a high predicted value like .80 for the positive class, or does it show a poor ability to recognize the positive class and assign a lower predicted value like .50? Logloss ranges between 0 and 1, with 0 meaning that the model correctly assigns a probability of 0% or 100%.

Binary classification equation:

\[
Logloss = - \frac{1}{N} \sum_{i=1}^{N} w_i (y_i \ln(p_i) + (1 - y_i) \ln(1 - p_i))
\]

Multiclass classification equation:

\[
Logloss = - \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{C} w_i (y_{i,j} \ln(p_{i,j}))
\]

Where:
- \(N\) is the total number of rows (observations) of your corresponding dataframe.
- \(w\) is the per row user-defined weight (defaults is 1).
- \(C\) is the total row number of classes (C=2 for binary classification).
- \(p\) is the predicted value (uncalibrated probability) assigned to a given row (observation).
- \(y\) is the actual target value.

- **AUC** (Area Under the Receiver Operating Characteristic Curve): This model metric is used to evaluate how well a binary classification model is able to distinguish between true positives and false positives. For multi-class problems, this score is computed by micro-averaging the ROC curves for each class. Use MACROAUC if you prefer the macro average.

An AUC of 1 indicates a perfect classifier, while an AUC of .5 indicates a poor classifier whose performance is no better than random guessing.

- **AUCPR** (Area Under the Precision-Recall Curve): This model metric is used to evaluate how well a binary classification model is able to distinguish between precision recall pairs or points. These values are obtained using different thresholds on a probabilistic or other continuous-output classifier. AUCPR is an average of the precision-recall weighted by the probability of a given threshold.

The main difference between AUC and AUCPR is that AUC calculates the area under the ROC curve and AUCPR calculates the area under the Precision Recall curve. The Precision Recall curve does not care about True Negatives. For imbalanced data, a large quantity of True Negatives usually overshadows the effects of changes in other metrics like False Positives. The AUCPR will be much more sensitive to True Positives, False Positives, and False Negatives than AUC. As such, AUCPR is recommended over AUC for highly imbalanced data.
• **MACROAUC** (Macro Average of Areas Under the Receiver Operating Characteristic Curves): For multiclass classification problems, this score is computed by macro-averaging the ROC curves for each class (one per class). The area under the curve is a constant. A MACROAUC of 1 indicates a perfect classifier, while a MACROAUC of .5 indicates a poor classifier whose performance is no better than random guessing. This option is not available for binary classification problems.

**Scorer Best Practices - Regression**

When deciding which scorer to use in a regression problem, some main questions to ask are:

- Do you want your scorer sensitive to outliers?
- What unit should the scorer be in?

**Sensitive to Outliers**

Certain scorers are more sensitive to outliers. When a scorer is sensitive to outliers, it means that it is important that the model predictions are never “very” wrong. For example, let’s say we have an experiment predicting number of days until an event. The graph below shows the absolute error in our predictions.

![Graph showing absolute error in days](image)

Usually our model is very good. We have an absolute error less than 1 day about 70% of the time. There is one instance, however, where our model did very poorly. We have one prediction that was 30 days off.

Instances like this will more heavily penalize scorers that are sensitive to outliers. If we do not care about these outliers in poor performance as long as we typically have a very accurate prediction, then we would want to select a scorer that is robust to outliers. We can see this reflected in the behavior of the scorers: **MSE** and **RMSE**.

<table>
<thead>
<tr>
<th></th>
<th>MSE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlier</td>
<td>0.99</td>
<td>2.64</td>
</tr>
<tr>
<td>No Outlier</td>
<td>0.80</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Calculating the **RMSE** and **MSE** on our error data, the **RMSE** is more than twice as large as the **MSE** because **RMSE** is sensitive to outliers. If we remove the one outlier record from our calculation, **RMSE** drops down significantly.
Performance Units

Different scorers will show the performance of the Driverless AI experiment in different units. Let’s continue with our example where our target is to predict the number of days until an event. Some possible performance units are:

- **Same as target**: The unit of the scorer is in days
  - ex: MAE = 5 means the model predictions are off by 5 days on average
- **Percent of target**: The unit of the scorer is the percent of days
  - ex: MAPE = 10% means the model predictions are off by 10 percent on average
- **Square of target**: The unit of the scorer is in days squared
  - ex: MSE = 25 means the model predictions are off by 5 days on average (square root of 25 = 5)

Comparison

<table>
<thead>
<tr>
<th>Metric</th>
<th>Units</th>
<th>Sensitive to Outliers</th>
<th>Tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>R^2</td>
<td>scaled between 0 and 1</td>
<td>No</td>
<td>use when you want performance scaled between 0 and 1</td>
</tr>
<tr>
<td>MSE</td>
<td>square of target</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>same as target</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>RM-SLE</td>
<td>log of target</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>RM-SPE</td>
<td>percent of target</td>
<td>Yes</td>
<td>use when target values are across different scales</td>
</tr>
<tr>
<td>MAE</td>
<td>same as target</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>MAPE</td>
<td>percent of target</td>
<td>No</td>
<td>use when target values are across different scales</td>
</tr>
<tr>
<td>SMAPE</td>
<td>percent of target divided by 2</td>
<td>No</td>
<td>use when target values close to 0</td>
</tr>
</tbody>
</table>

Scorer Best Practices - Classification

When deciding which scorer to use in a classification problem some main questions to ask are:

- Do you want the scorer to evaluate the predicted probabilities or the classes that those probabilities can be converted to?
- Is your data imbalanced?

Scorer Evaluates Probabilities or Classes

The final output of a Driverless AI model is a predicted probability that a record is in a particular class. The scorer you choose will either evaluate how accurate the probability is or how accurate the assigned class is from that probability.

Choosing this depends on the use of the Driverless AI model. Do we want to use the probabilities or do we want to convert those probabilities into classes? For example, if we are predicting whether a customer will churn, we may take the predicted probabilities and turn them into classes - customers who will churn vs customers who won’t churn. If we are predicting the expected loss of revenue, we will instead use the predicted probabilities (predicted probability of churn * value of customer).

If your use case requires a class assigned to each record, you will want to select a scorer that evaluates the model’s performance based on how well it classifies the records. If your use case will use the probabilities, you will want to select a scorer that evaluates the model’s performance based on the predicted probability.

Robust to Imbalanced Data
For certain use cases, positive classes may be very rare. In these instances, some scorers can be misleading. For example, if I have a use case where 99% of the records have \texttt{Class = No}, then a model which always predicts \texttt{No} will have 99% accuracy. For these use cases, it is best to select a metric that does not include True Negatives or considers relative size of the True Negatives like AUCPR or MCC.

### Comparison

<table>
<thead>
<tr>
<th>Metric</th>
<th>Evaluation Based On</th>
<th>Tip</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCC</td>
<td>Class</td>
<td>good for imbalanced data</td>
</tr>
<tr>
<td>F1</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>F0.5</td>
<td>Class</td>
<td>good when you want to give more weight to precision</td>
</tr>
<tr>
<td>F2</td>
<td>Class</td>
<td>good when you want to give more weight to recall</td>
</tr>
<tr>
<td>Accuracy</td>
<td>Class</td>
<td>highly interpretable</td>
</tr>
<tr>
<td>Logloss</td>
<td>Probability</td>
<td></td>
</tr>
<tr>
<td>AUC</td>
<td>Class</td>
<td></td>
</tr>
<tr>
<td>AUCPR</td>
<td>Class</td>
<td>good for imbalanced data</td>
</tr>
</tbody>
</table>

#### 13.2 New Experiments

1. Run an experiment by selecting [Click for Actions] button beside the dataset that you want to use. Click Predict to begin an experiment.

2. The Experiment Settings form displays and auto-fills with the selected dataset. Optionally specify a validation dataset and/or a test dataset.
   - The validation set is used to tune parameters (models, features, etc.). If a validation dataset is not provided, the training data is used (with holdout splits). If a validation dataset is provided, training data is not used for parameter tuning - only for training. A validation dataset can help to improve the generalization performance on shifting data distributions.
• The test dataset is used for the final stage scoring and is the dataset for which model metrics will be computed against. Test set predictions will be available at the end of the experiment. This dataset is not used during training of the modeling pipeline.

Keep in mind that these datasets must have the same number of columns as the training dataset. Also note that if provided, the validation set is not sampled down, so it can lead to large memory usage, even if accuracy=1 (which reduces the train size).

3. Specify the target (response) column. Note that not all explanatory functionality will be available for multiclass classification scenarios (scenarios with more than two outcomes). When the target column is selected, Driverless AI automatically provides the target column type and the number of rows. If this is a classification problem, then the UI shows unique and frequency statistics (Target Freq/Most Freq) for numerical columns. If this is a regression problem, then the UI shows the dataset mean and standard deviation values.

Notes Regarding Frequency:

• For data imported in versions <= 1.0.19, TARGET FREQ and MOST FREQ both represent the count of the least frequent class for numeric target columns and the count of the most frequent class for categorical target columns.

• For data imported in versions 1.0.20-1.0.22, TARGET FREQ and MOST FREQ both represent the frequency of the target class (second class in lexicographic order) for binomial target columns; the count of the most frequent class for categorical multinomial target columns; and the count of the least frequent class for numeric multinomial target columns.

• For data imported in version 1.0.23 (and later), TARGET FREQ is the frequency of the target class for binomial target columns, and MOST FREQ is the most frequent class for multinomial target columns.

4. The next step is to set the parameters and settings for the experiment. (Refer to the Experiment Settings section for more information about these settings.) You can set the parameters individually, or you can let Driverless AI infer the parameters and then override any that you disagree with. Available parameters and settings include the following:

• Dropped Columns: The columns we do not want to use as predictors such as ID columns, columns with data leakage, etc.

• Weight Column: The column that indicates the per row observation weights. If “None” is specified, each row will have an observation weight of 1.

• Fold Column: The column that indicates the fold. If “None” is specified, the folds will be determined by Driverless AI. This is set to “Disabled” if a validation set is used.

• Time Column: The column that provides a time order, if applicable. If “AUTO” is specified, Driverless AI will auto-detect a potential time order. If “OFF” is specified, auto-detection is disabled. This is set to “Disabled” if a validation set is used.

• Specify the scorer to use for this experiment. The scorers vary based on whether this is a classification or regression experiment. Available scorers include:

  – Regression: GINI, R2, MSE, RMSE (default), RMSLE, RMSPE, MAE, MER, MAPE, SMAPE
  – Classification: GINI, MCC, F05, F1, F2, ACCURACY, LOGLOSS, AUC (default), AUCPR

• Desired relative Accuracy from 1 to 10

• Desired relative Time from 1 to 10

• Desired relative Interpretability from 1 to 10

Driverless AI will automatically infer the best settings for Accuracy, Time, and Interpretability and provide you with an experiment preview based on those suggestions. If you adjust these knobs, the experiment preview will automatically update based on the new settings.
Expert Settings (optional):

- Optionally specify additional expert settings for the experiment. Refer to the *Expert Settings* section for more information about these settings. The default values for these options are derived from the environment variables in the config.toml file. Refer to the *Setting Environment Variables* section for more information.

Additional settings (optional):

- **Classification** or **Regression** button. Driverless AI automatically determines the problem type based on the response column. Though not recommended, you can override this setting by clicking this button.

- **Reproducible**: This button allows you to build an experiment with a random seed and get reproducible results. If this is disabled (default), then results will vary between runs.

- **Enable GPUs**: Specify whether to enable GPUs. (Note that this option is ignored on CPU-only systems.)
5. Click **Launch Experiment** to start the experiment.

The experiment launches with a randomly generated experiment name. You can change this name at anytime during or after the experiment. Mouse over the name of the experiment to view an edit icon, then type in the desired name.

As the experiment runs, a running status displays in the upper middle portion of the UI. First Driverless AI figures out the backend and determines whether GPUs are running. Then it starts parameter tuning, followed by feature engineering. Finally, Driverless AI builds the scoring pipeline.

In addition to the status, the UI also displays:

- Details about the dataset.
- The iteration data (internal validation) for each cross validation fold along with the specified scorer value. Click on a specific iteration or drag to view a range of iterations. Double click in the graph to reset the view. In this graph, each “column” represents one iteration of the experiment. During the iteration, Driverless AI will train \( n \) models. (This is called individuals in the experiment preview.) So for any column, you may see the score value for those \( n \) models for each iteration on the graph.
- The variable importance values. To view variable importance for a specific iteration, just select that iteration in the Iteration Data graph. The Variable Importance list will automatically update to show variable importance information for that iteration. Hover over an entry to view more info. **Note:** When hovering over an entry, you may notice the term “Internal[…] specification.” This label is used for features that do not need to be translated/explained and ensures that all features are uniquely identified.
• CPU/Memory information including Notifications, Logs, and Trace info.

For classification problems, the lower right section includes a toggle between an ROC curve, Precision-Recall graph, Lift chart, Gains chart, and GPU Usage information (if GPUs are available). For regression problems, the lower right section includes a toggle between an Actual vs. Predicted chart and GPU Usage information (if GPUs are available). (Refer to the Experiment Graphs section for more information.) Upon completion, an Experiment Summary section will populate in the lower right section.

The bottom portion of the experiment screen will show any warnings that Driverless AI encounters. You can hide this pane by clicking the x icon.

You can stop experiments that are currently running. Click the Finish button to stop the experiment. This jumps the experiment to the end and completes the ensembling and the deployment package. You can also click Abort to terminate the experiment. (You will be prompted to confirm the abort.) Aborted experiments will display on the Experiments page as Failed. You can restart aborted experiments by clicking the right side of the experiment, then selecting Restart from Last Checkpoint. This will start a new experiment based on the aborted one. Alternatively, you can start a new experiment based on the aborted one by selecting New Model with Same Params. Refer to checkpointing for more information.

13.3 Experiment Graphs

This section describes the dashboard graphs that display for running and completed experiments. These graphs are interactive. Hover over a point on the graph for more details about the point.

13.3.1 Binary Classification Experiments

For Binary Classification experiments, Driverless AI shows a ROC Curve, a Precision-Recall graph, a Lift chart, a Kolmogorov-Smirnov chart, and a Gains chart.
**ROC**: This shows Receiver-Operator Characteristics curve stats on validation data along with the best Accuracy, FCC, and F1 values.

The area under this curve is called AUC. The True Positive Rate (TPR) is the relative fraction of correct positive predictions, and the False Positive Rate (FPR) is the relative fraction of incorrect positive corrections. Each point corresponds to a classification threshold (e.g., YES if probability $\geq 0.3$ else NO). For each threshold, there is a unique confusion matrix that represents the balance between TPR and FPR. Most useful operating points are in the top left corner in general.

Hover over a point in the ROC curve to see the True Positive, True Negative, False Positive, False Negative, Threshold, FPR, TPR, Accuracy, F1, and MCC value for that point.

If a test set was provided for the experiment, then click on the **Validation Metrics** button below the graph to view these stats on test data.

**Precision-Recall**: This shows the Precision-Recall curve on validation data along with the best Accuracy, FCC, and F1 values. The area under this curve is called AUCPR.

- **Precision**: correct positive predictions (TP) / all positives (TP + FP).
- **Recall**: correct positive predictions (TP) / positive predictions (TP + FN).

Each point corresponds to a classification threshold (e.g., YES if probability $\geq 0.3$ else NO). For each threshold, there is a unique confusion matrix that represents the balance between Recall and Precision. This ROCPR curve can be more insightful than the ROC curve for highly imbalanced datasets.

Hover over a point in this graph to see the True Positive, True Negative, False Positive, False Negative, Threshold, Recall, Precision, Accuracy, F1, and MCC value for that point.

If a test set was provided for the experiment, then click on the **Validation Metrics** button below the graph to view these stats on test data.

**Lift**: This chart shows lift stats on validation data. For example, “How many times more observations of the positive target class are in the top predicted 1%, 2%, 10%, etc. (cumulative) compared to selecting observations randomly?” By definition, the Lift at 100% is 1.0.

Hover over a point in the Lift chart to view the quantile percentage and cumulative lift value for that point.

If a test set was provided for the experiment, then click on the **Validation Metrics** button below the graph to view these stats on test data.
• **Kolmogorov-Smirnov**: This chart measures the degree of separation between positives and negatives for validation or test data.

Hover over a point in the chart to view the quantile percentage and Kolmogorov-Smirnov value for that point.

If a test set was provided for the experiment, then click on the **Validation Metrics** button below the graph to view these stats on test data.

• **Gains**: This shows Gains stats on validation data. For example, “What fraction of all observations of the positive target class are in the top predicted 1%, 2%, 10%, etc. (cumulative)?” By definition, the Gains at 100% are 1.0.

Hover over a point in the Gains chart to view the quantile percentage and cumulative gain value for that point.

If a test set was provided for the experiment, then click on the **Validation Metrics** button below the graph to view these stats on test data.

### 13.3.2 Multiclass Classification Experiments

The ROC Curve, Precision-Recall graph, Lift chart, Kolmogorov-Smirnov chart, and Gains chart are also shown for multiclass problems. Driverless AI does this by considering the multi-class problem as multiple one-vs-all problems. This method is known as micro-averaging (reference: http://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#multiclass-settings).

For example, you may want to predict the species in the iris data. The predictions would look something like this:

<table>
<thead>
<tr>
<th>class.Iris-setosa</th>
<th>class.Iris-versicolor</th>
<th>class.Iris-virginica</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9628</td>
<td>0.021</td>
<td>0.0158</td>
</tr>
<tr>
<td>0.0182</td>
<td>0.3172</td>
<td>0.6646</td>
</tr>
<tr>
<td>0.0191</td>
<td>0.9534</td>
<td>0.0276</td>
</tr>
</tbody>
</table>

To create these charts, Driverless AI converts the results to 3 one-vs-all problems:

<table>
<thead>
<tr>
<th>prob-setosa</th>
<th>actual-setosa</th>
<th>prob-versicolor</th>
<th>actual-versicolor</th>
<th>prob-virginica</th>
<th>actual-virginica</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.9628</td>
<td>0</td>
<td>0.021</td>
<td>0</td>
<td>0.0158</td>
<td>0</td>
</tr>
<tr>
<td>0.0182</td>
<td>0</td>
<td>0.3172</td>
<td>1</td>
<td>0.6646</td>
<td>0</td>
</tr>
<tr>
<td>0.0191</td>
<td>0</td>
<td>0.9534</td>
<td>1</td>
<td>0.0276</td>
<td>0</td>
</tr>
</tbody>
</table>

The result is 3 vectors of predicted and actual values for binomial problems. Driverless AI concatenates these 3 vectors together to compute the charts.

```
predicted = [0.9628, 0.0182, 0.0191, 0.021, 0.3172, 0.9534, 0.0158, 0.6646, 0.0276]
actual = [1, 0, 0, 0, 1, 1, 0, 0, 0]
```

### Multiclass Confusion Matrix

Hover over a point in the ROC or Precision Recall graph to view a Multiclass Confusion Matrix for that point (shown against the validation set). For each threshold, the confusion matrix represents the balance between TPR and FPR (ROC) or Precision and Recall (Prec-Recall). In general, most useful operating points are in the top left corner.
13.3.3 Regression Experiments

An Actual vs. Predicted table is available for Regression experiments. This shows Actual vs Predicted values on validation data. A small sample of values are displayed. A perfect model has a diagonal line.

Hover over a point on the graph to view the Actual and Predicted values for that point.

If a test set was provided for the experiment, then click on the Validation Metrics button below the graph to view these stats on test data.
13.4 Completed Experiment

After an experiment status changes from RUNNING to COMPLETE, the UI provides you with several options:

- **Deploy**: Refer to *Deploying the MOJO Pipeline*. (Only available after building MOJO scoring pipelines. Not available for PPC64LE environments.)

- **Interpret this Model**: Refer to *Interpreting a Model*. (Not available for NLP experiments.)

- **Diagnose Model on New Dataset**: Refer to *Diagnosing a Model*.

- **Score on Another Dataset**: Refer to *Score on Another Dataset*.

- **Transform Another Dataset**: Refer to *Transform Another Dataset*. (Not available for Time Series experiments.)

- **Download Predictions dropdown**:
  - Training (Holdout) Predictions: In csv format, available if a validation set was NOT provided.
  - Validation Set Predictions: In csv format, available if a validation set was provided.
  - Test Set Predictions: In csv format, available if a test dataset is used.

- **Download Python Scoring Pipeline**: A standalone Python scoring pipeline for H2O Driverless AI. Refer to *Driverless AI Standalone Python Scoring Pipeline*.

- **Build MOJO Scoring Pipeline**: A standalone Model Object, Optimized scoring pipeline. Refer to *Driverless AI MOJO Scoring Pipeline*. (Not available for TensorFlow, RuleFit, or FTRL models.)

- **Download Experiment Summary**: A zip file containing the following files. Refer to the *Experiment Summary* section for more information.
  - A summary of the experiment
  - The experiment features along with their relative importance
  - Ensemble information
  - An experiment preview
13.4.1 Experiment Summary

An experiment summary is available for each completed experiment. Click the Download Experiment Summary button to download the h2oai_experiment_summary_<experiment>.zip file.
The files within the experiment summary zip provide textual explanations of the graphical representations that are shown on the Driverless AI UI. Details of each artifact are described below.

**Experiment Report**

A report file is included in the experiment summary. This report provides insight into the training data and any detected shifts in distribution, the validation schema selected, model parameter tuning, feature evolution and the final set of features chosen during the experiment.

- **report.docx**: the report available in Word format

Click here to download and view a sample experiment report in Word format.

**Experiment Overview Artifacts**

The Experiment Summary contains artifacts that provide overviews of the experiment.

- **preview.txt**: Provides a preview of the experiment. (This is the same information that was included on the UI before starting the experiment.)
- **summary.txt**: Provides the same summary that appears in the lower-right portion of the UI for the experiment.

**Tuning Artifacts**

During the Driverless AI experiment, model tuning is performed to determined the optimal algorithm and parameter settings for the provided dataset. For regression problems, target tuning is also performed to determine the best way
Using Driverless AI, Release 1.6.2

to represent the target column (i.e. does taking the log of the target column improve results). The results from these
tuning steps are available in the Experiment Summary.

- **tuning_leaderboard**: A table of the model tuning performed along with the score generated from the model
  and training time. (Available in txt or json.)

- **target_transform_tuning_leaderboard.txt**: A table of the transforms applied to the target column along with
  the score generated from the model and training time. (This will be empty for binary and multiclass use cases.)

**Features Artifacts**

Driverless AI performs feature engineering on the dataset to determine the optimal representation of the data. The
top features used in the final model can be seen in the GUI. The complete list of features used in the final model is
available in the Experiment Summary artifacts.

The Experiment Summary also provides a list of the original features and their estimated feature importance. For example, given the features in the final Driverless AI model, we can estimate the feature importance of the original
features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Feature Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>NumToCatWoE:PAY_AMT2</td>
<td>1</td>
</tr>
<tr>
<td>PAY_3</td>
<td>0.92</td>
</tr>
<tr>
<td>ClusterDist9:BILL_AMT1:LIMIT_BAL:PAY_3</td>
<td>0.90</td>
</tr>
</tbody>
</table>

To calculate the feature importance of PAY_3, we can aggregate the feature importance for all variables that used PAY_3:

- **NumToCatWoE:PAY_AMT2**: 1 * 0 (PAY_3 not used.)
- **PAY_3**: 0.92 * 1 (PAY_3 is the only variable used.)
- **ClusterDist9:BILL_AMT1:LIMIT_BAL:PAY_3**: 0.90 * 1/3 (PAY_3 is one of three variables used.)

Estimated Feature Importance = (1*0) + (0.92*1) + (0.9*(1/3)) = 1.22

**Note**: The feature importance is converted to relative feature importance. (The feature with the highest estimated
feature importance will have a relative feature importance of 1).

- **features**: A complete list of all features used in the final model, a description of the feature, and the relative
  feature importance. (Available in txt or json.)

- **features_orig**: A list of the original features provided and an estimate of the relative feature importance of that
  original feature in the final model. (Available in txt or json.)

**Final Model Artifacts**

The Experiment Summary includes artifacts that describe the final model. This is the model that is used to score new
datasets and create the MOJO scoring pipeline. The final model may be an ensemble of models depending on the
Accuracy setting.

- **ensemble.txt**: A summary of the final model which includes a description of the model(s), gains/lifts table,
  confusion matrix, and scores of the final model for our list of scorers.

- **ensemble_description.txt**: A sentence describing the final model. (For example: Final TensorFlowModel
  pipeline with ensemble_level=0 transforming 21 original features -> 54 features in each of 1 models each fit on
  full training data (i.e. no hold-out).)

- **ensemble_model_description.json**: A json file describing the model(s) and for ensembles how the model pre-
  dictions are weighted.
• **ensemble_model_params.json**: A json file describing the parameters of the model(s).

• **ensemble_folds_data.json**: A json file describing the folds used for the final model(s). This includes the size of each fold of data and the performance of the final model on each fold.

The Experiment Summary also includes artifacts about the final model performance.

• **ensemble_scores.json**: The scores of the final model for our list of scorers.

• **ensemble_confusion_matrix**: The confusion matrix for the internal validation and test data if test data is provided.

• **ensemble_confusion_matrix_stats_test.json**: Confusion matrix statistics on the test data. (Only available if test data provided)

• **ensemble_gains**: The lift and gains table for the internal validation and test data if test data is provided. (Visualization of lift and gains can be seen in the UI.)

• **ensemble_roc**: The ROC and Precision Recall table for the internal validation and test data if test data is provided. (Visualization of ROC and Precision Recall curve can be seen in the UI.)

### 13.5 Viewing Experiments

The upper-right corner of the Driverless AI UI includes an **Experiments** link.

Click this link to open the Experiments page. From this page, you can rename an experiment, view previous experiments, begin a new experiment, rerun an experiment, and delete an experiment.

![Experiments Screen](image)

**13.5.1 Checkpointing, Rerunning, and Retraining**

In Driverless AI, you can retry an experiment from the last checkpoint, you can run a new experiment using an existing experiment’s settings, and you can retrain an experiment’s final pipeline.
Checkpointing Experiments

In real-world scenarios, data can change. For example, you may have a model currently in production that was built using 1 million records. At a later date, you may receive several hundred thousand more records. Rather than building a new model from scratch, Driverless AI includes H2O.ai Brain, which enables caching and smart re-use of prior models to generate features for new models.

You can configure one of the following Brain levels in the experiment’s Expert Settings.

-1: Don’t use any brain cache

0: Don’t use any brain cache but still write to cache

1: Smart checkpoint if an old experiment_id is passed in (for example, via running “resume one like this” in the GUI)

2: Smart checkpoint if the experiment matches all column names, column types, classes, class labels, and time series options identically. (default)

3: Smart checkpoint like level #1, but for the entire population. Tune only if the brain population is of insufficient size.

4: Smart checkpoint like level #2, but for the entire population. Tune only if the brain population is of insufficient size.

5: Smart checkpoint like level #4, but will scan over the entire brain cache of populations (starting from resumed experiment if chosen) in order to get the best scored individuals.

If you choose Level 2 (default), then Level 1 is also done when appropriate.

To make use of smart checkpointing, be sure that the new data has:

- The same data column names as the old experiment
- The same data types for each column as the old experiment. (This won’t match if, e.g., a column was all int and then had one string row.)
- The same target as the old experiment
- The same target classes (if classification) as the old experiment
- For time series, all choices for intervals and gaps must be the same

When the above conditions are met, then you can:
Start the same kind of experiment, just rerun for longer.

Use a smaller or larger data set (i.e. fewer or more rows).

Effectively do a final ensemble re-fit by varying the data rows and starting an experiment with a new accuracy, time=1, and interpretability. Check the experiment preview for what the ensemble will be.

Restart/Resume a cancelled, aborted, or completed experiment

To run smart checkpointing on an existing experiment, click the right side of the experiment that you want to retry, then select **Restart from Last Checkpoint**. The experiment settings page opens. Specify the new dataset. If desired, you can also change experiment settings, though the target column must be the same. Click **Launch Experiment** to resume the experiment from the last checkpoint and build a new experiment.

The smart checkpointing continues by adding a prior model as another model used during tuning. If that prior model is better (which is likely if it was run for more iterations), then that smart checkpoint model will be used during feature evolution iterations and final ensemble.

Notes:

- Driverless AI does not guarantee exact continuation, only smart continuation from any last point.

- The directory where the H2O.ai Brain meta model files are stored is `tmp/H2O.ai_brain`. In addition, the default maximum brain size is 20GB. Both the directory and the maximum size can be changed in the config.toml file.

**Rerunning Experiments**

To run a new experiment using an existing experiment’s settings, click the right side of the experiment that you want to use as the basis for the new experiment, then select **New Model with Same Params**. This opens the experiment settings page. From this page, you can rerun the experiment using the original settings, or you can specify to use new data and/or specify different experiment settings. Click **Launch Experiment** to create a new experiment with the same options.

**Retrain Final Pipeline**

To retrain an experiment’s final pipeline, click the right side of the experiment that you want to use as the basis for the new experiment, then select **Retrain Final Pipeline**. This opens the experiment settings page with the same settings as the original experiment except that Time is set to 0.

**13.5.2 Deleting Experiments**

To delete an experiment, click the right side of the experiment that you want to remove, then select **Delete**. A confirmation message will display asking you to confirm the delete. Click **OK** to delete the experiment or **Cancel** to return to the experiments page without deleting.
### Experiments

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Target</th>
<th>Validation Score</th>
<th>Test Score</th>
<th>Score</th>
<th>Size</th>
<th>Accuracy</th>
<th>Time</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>H20.ai</td>
<td>Default payment</td>
<td>0.78440</td>
<td>0.76985</td>
<td>AUC</td>
<td>234MB</td>
<td>7</td>
<td>3</td>
<td>Done</td>
</tr>
<tr>
<td>r.numele</td>
<td>Weekly Sales</td>
<td>4963.7</td>
<td>NA</td>
<td>RMSE</td>
<td>261MB</td>
<td>4</td>
<td>3</td>
<td>Done</td>
</tr>
</tbody>
</table>

- **New Model with Same Params**
- **Restart from Last Checkpoint**
- **Retrain Final Pipeline**
- **Delete**
DIAGNOSING A MODEL

The **Diagnosing Model on New Dataset** option allows you to view model performance for multiple scorers based on existing model and dataset.

On the completed experiment page, click the **Diagnose Model on New Dataset** button.

**Note:** You can also diagnose a model by selecting **Diagnostic** from the top menu, then selecting an experiment and test dataset.

Select a dataset to use when diagnosing this experiment. Note that the dataset must include the target column that is in the original dataset. At this point, Driverless AI will begin calculating all available scores for the experiment.

When the diagnosis is complete, it will be available on the **Model Diagnostics** page. Click on the new diagnosis. From this page, you can download predictions. You can also view scores and metric plots. The plots are interactive. Click a graph to enlarge. In the enlarged view, you can hover over the graph to view details for a specific point. You can also download the graph in the enlarged view.
Classification Metric Plots

Classification metric plots include the following graphs:

- ROC Curve
- Precision-Recall Curve
- Cumulative Gains
- Lift Chart
- Kolmogorov-Smirnov Chart
- Confusion Matrix

Regression Metric Plots

Regression metric plots include the following graphs:

- Actual vs Predicted
- Residual Plot with LOESS curve
- Residual Histogram
There are two methods you can use for interpreting models:

- Using the **Interpret this Model** button on a completed experiment page to interpret a Driverless AI model on original and transformed features.

- Using the **MLI** (Machine Learning Interpretability) link in the upper right corner of the UI to interpret either a Driverless AI model or an external model.

**Notes:**

- MLI no longer runs on experiments from previous releases, but also does not require internet to run on current models.
- MLI is not available for NLP experiments.

### 15.1 Additional Resources

- Click here to download our MLI cheat sheet.
- Click here to view our H2O Driverless AI Machine Learning Interpretability walkthrough video.

### 15.2 Intrepret this Model Button - Non-Time-Series

Clicking the **Interpret this Model** button on a completed experiment page launches the Model Interpretation for that experiment. Python and Java logs can be viewed for non-time-series experiments while the interpretation is running.

For non-time-series experiments, this page provides several visual explanations and reason codes for the trained Driverless AI model and its results. More information about this page is available in the **Understanding the Model Interpretation Page** section later in this chapter.
15.3 Interpret this Model Button - Time-Series

Limitations:

- MLI for time series will not run without a test set from a Driverless AI experiment. In future releases, internal validation predictions will be used if a test set is not provided.

- When the test set contains actuals, you will see the time series metric plot and the group metrics table. If there are no actuals, MLI will run, but you will see only the prediction value time series and a Shapley table.

- Model interpretation for time-series experiments is currently in an alpha state and changes are imminent.

15.3.1 Multi-Group Time Series MLI

This section describes how to run MLI on time series data for multiple groups.

1. Click the Interpret this Model (alpha) button on a completed time series experiment to launch Model Interpretation for that experiment. This page includes the following:

   - A Help panel describing how to read and use this page. Click the Hide Help Button to hide this text.

   - If the test set includes actuals, then a panel will display showing a time series plot and the top and bottom group matrix tables based on the scorer that was used in the experiment. Note that this panel can be resized if necessary.

   - A Show Summary button that provides details about the experiment settings that were used.

   - A Group Search entry field for selecting the groups to view.
2. Scroll to the bottom of the panel and select a grouping in the **Group Search** field to view a graph of Actual vs. Predicted values for the group. The outputted graph can be downloaded to your local machine.

3. Click on a prediction point in the plot (white line) to view Shapley values for that prediction point. The Shapley values plot can also be downloaded to your local machine.
4. Click **Add Panel** to add a new MLI Time Series panel. This allows you to compare different groups in the same model and also provides the flexibility to do a “side-by-side” comparison between different models.

### 15.3.2 Single Time Series MLI

Time Series MLI can also be run when only one group is available.

1. Click the **Interpret this Model (alpha)** button on a completed time series experiment to launch Model Interpretation for that experiment. This page includes the following:

   - A Help panel describing how to read and use this page. Click the **Hide Help Button** to hide this text.
   - If the test set includes actuals, then a panel will display showing a time series plot showing the single group’s matrix tables based on the scorer that was used in the experiment. Note that this panel can be resized if necessary.
   - A **Show Summary** button that provides details about the experiment settings that were used.
   - A **Group Search** entry field for selecting the group to view. Note that for Single Time Series MLI, there will only be one option in this field.
2. Scroll to the bottom of the panel and select an option in the **Group Search** field to view a graph of Actual vs. Predicted values for the group. (Note that for Single Time Series MLI, there will only be one option in this field.) The outputted graph can be downloaded to your local machine.

![Actual vs Predicted Graph](image)

3. Click on a prediction point in the plot (white line) to view Shapley values for that prediction point. The Shapley values plot can also be downloaded to your local machine.

![Shapley Values Plot](image)

4. Click **Add Panel** to add a new MLI Time Series panel. This allows you to do a “side-by-side” comparison between different models.

### 15.4 Model Interpretation on Driverless AI Models

This method allows you to run model interpretation on a Driverless AI model. This method is similar to clicking “Interpret This Model” on an experiment summary page.

1. Click the **MLI** link in the upper-right corner of the UI to view a list of interpreted models.
2. Click the **New Interpretation** button.

3. Select the dataset that was used to train the model that you will use for interpretation.

4. Specify the Driverless AI model that you want to use for the interpretation. Once selected, the Target Column used for the model will be selected.

5. Select a LIME method of either K-LIME (default) or LIME-SUP.
   - **K-LIME** creates one global surrogate GLM on the entire training data and also creates numerous local surrogate GLMs on samples formed from \(k\)-means clusters in the training data. The features used for \(k\)-means are selected from the Random Forest surrogate model’s variable importance. The number of features used for \(k\)-means is the minimum of the top 25% of variables from the Random Forest surrogate model’s variable importance and the max number of variables that can be used for \(k\)-means, which is set by the user in the config.toml setting for `mli_max_number_cluster_vars` (Note, if the number of features in the dataset are less than or equal to 6, then all features are used for \(k\)-means clustering.) The previous setting can be turned off to use all features for \(k\)-means by setting `use_all_columns_klime_kmeans` in the config.toml file to true. All penalized GLM surrogates are trained to model the predictions of the Driverless AI model. The number of clusters for local explanations is chosen by a grid search in which the \(R^2\) between the Driverless AI model predictions and all of the local K-LIME model predictions is maximized. The global and local linear model’s intercepts, coefficients, \(R^2\) values, accuracy, and predictions can all be used to debug and develop explanations for the Driverless AI model’s behavior.
   - **LIME-SUP** explains local regions of the trained Driverless AI model in terms of the original variables. Local regions are defined by each leaf node path of the decision tree surrogate model instead of simulated, perturbed observation samples - as in the original LIME. For each local region, a local GLM model is trained on the original inputs and the predictions of the Driverless AI model. Then the parameters of this local GLM can be used to generate approximate, local explanations of the Driverless AI model.

6. For K-LIME interpretations, specify the depth that you want for your decision tree surrogate model. The tree depth value can be a value from 2-5 and defaults to 3. For LIME-SUP interpretations, specify the LIME-SUP tree depth. This can be a value from 2-5 and defaults to 3.

7. Specify whether to use original features or transformed features in the surrogate model for the new interpretation.
   - **Note:** If this option is disabled, then the K-LIME clustering column option will not be available, and quantile binning will not be available.

8. Specify whether to perform the interpretation on a sample of the training data. By default, MLI will sample the training dataset if it is greater than 100k rows. (Note that this value can be modified in the config.toml setting for `mli_sample_size`.) Turn this toggle off to run MLI on the entire dataset.

9. Optionally specify weight and dropped columns.

10. For K-LIME interpretations, optionally specify a clustering column. Note that this column should be categorical. Also note that this is only available when K-LIME is used as the LIME method and when **Use Original Features** is enabled. If the LIME method is changed to LIME-SUP, then this option is no longer available.

11. Optionally specify the number of surrogate cross-validation folds to use (from 0 to 10). When running experiments, Driverless AI automatically splits the training data and uses the validation data to determine the
performance of the model parameter tuning and feature engineering steps. For a new interpretation, Driverless
AI uses 3 cross-validation folds by default for the interpretation.

12. For K-LIME interpretations, optionally specify one or more columns to generate decile bins (uniform distribu-
tion) to help with MLI accuracy. Columns selected are added to top $n$ columns for quantile binning selection. If a column is not numeric or not in the dataset (transformed features), then the column will be skipped. **Note:** This option is only available when **Use Original Features** is enabled.

13. For K-LIME interpretations, optionally specify the number of top variable importance numeric columns to run decile binning to help with MLI accuracy. (Note that variable importances are generated from a Random Forest model.) This defaults to 0, and the maximum value is 10. **Note:** This option is only available when **Use Original Features** is enabled.

14. Click the **Launch MLI** button.

---

**15.5 Model Interpretation on External Models**

Model Interpretation does not need to be run on a Driverless AI experiment. You can train an external model and run Model Interpretability on the predictions.

1. Click the **MLI** link in the upper-right corner of the UI to view a list of interpreted models.
2. Click the **New Interpretation** button.

3. Select the dataset that you want to use for the model interpretation. This must include a prediction column that was generated by the external model. If the dataset does not have predictions, then you can join the external predictions. An example showing how to do this in Python is available in the Run Model Interpretation on External Model Predictions section of the Credit Card Demo.

   **Note**: When running interpretations on an external model, leave the **Select Model** option empty. That option is for selecting a Driverless AI model.

4. Specify a Target Column (actuals) and the Prediction Column (scores from the model).

5. Select a LIME method of either **K-LIME** (default) or **LIME-SUP**.

   - **K-LIME** creates one global surrogate GLM on the entire training data and also creates numerous local surrogate GLMs on samples formed from $k$-means clusters in the training data. The features used for $k$-means are selected from the Random Forest surrogate model’s variable importance. The number of features used for $k$-means is the minimum of the top 25% of variables from the Random Forest surrogate model’s variable importance and the max number of variables that can be used for $k$-means, which is set by the user in the `config.toml` setting for `mli_max_number_cluster_vars`. (Note, if the number of features in the dataset are less than or equal to 6, then all features are used for $k$-means clustering.) The previous setting can be turned off to use all features for $k$-means by setting `use_all_columns_klime_kmeans` in the `config.toml` file to `true`. All penalized GLM surrogates are trained to model the predictions of the Driverless AI model. The number of clusters for local explanations is chosen by a grid search in which the $R^2$ between the Driverless AI model predictions and all of the local K-LIME model predictions is maximized. The global and local linear model’s intercepts, coefficients, $R^2$ values, accuracy, and predictions can all be used to debug and develop explanations for the Driverless AI model’s behavior.

   - **LIME-SUP** explains local regions of the trained Driverless AI model in terms of the original variables. Local regions are defined by each leaf node path of the decision tree surrogate model instead of simulated, perturbed observation samples - as in the original LIME. For each local region, a local GLM model is trained on the original inputs and the predictions of the Driverless AI model. Then the parameters of this local GLM can be used to generate approximate, local explanations of the Driverless AI model.

6. For K-LIME interpretations, specify the depth that you want for your decision tree surrogate model. The tree depth value can be a value from 2-5 and defaults to 3. For LIME-SUP interpretations, specify the LIME-SUP tree depth. This can be a value from 2-5 and defaults to 3.

7. Specify whether to perform the interpretation on a sample of the training data. By default, MLI will sample the training dataset if it is greater than 100k rows. (Note that this value can be modified in the `config.toml` setting for `mli_sample_size`.) Turn this toggle off to run MLI on the entire dataset.

8. Optionally specify weight and dropped columns.

9. For K-LIME interpretations, optionally specify a clustering column. Note that this column should be categorical. Also note that this is only available when K-LIME is used as the LIME method. If the LIME method is changed to LIME-SUP, then this option is no longer available.

10. Optionally specify the number of surrogate cross-validation folds to use (from 0 to 10). When running experiments, Driverless AI automatically splits the training data and uses the validation data to determine the
performance of the model parameter tuning and feature engineering steps. For a new interpretation, Driverless AI uses 3 cross-validation folds by default for the interpretation.

11. For K-LIME interpretations, optionally specify one or more columns to generate decile bins (uniform distribution) to help with MLI accuracy. Columns selected are added to top $n$ columns for quantile binning selection. If a column is not numeric or not in the dataset (transformed features), then the column will be skipped.

12. For K-LIME interpretations, optionally specify the number of top variable importance numeric columns to run decile binning to help with MLI accuracy. (Note that variable importances are generated from a Random Forest model.) This value is combined with any specific columns selected for quantile binning. This defaults to 0, and the maximum value is 10.

13. Click the **Launch MLI** button.

### 15.6 Understanding the Model Interpretation Page

The Model Interpretation page opens with a Summary of the interpretation. This page also provides left-hand navigation for viewing additional plots. This navigation includes:

- **Summary**: Provides a summary of the MLI experiment. See the Summary Page section for more information.
- **DAI Model**: Provides Feature Importance and Shapley plots for transformed features. (Shapley plots not supported for RuleFit and TensorFlow models.) See DAI Model Dropdown for more information.
- **Surrogate Models**: Provides KLIME and Decision Tree plots. This also includes a Random Forest submenu, which includes Global and Local Feature Importance plots for original features and a Partial Dependence plot.
Using Driverless AI, Release 1.6.2

See *Surrogate Models Dropdown* for more information.

- **Dashboard**: A single page with a Global Interpretable Model Explanations plot, a Feature Importance plot, a Decision Tree plot, and a Partial Dependence plot. See the *Dashboard Page* section for more information.
- **MLI Docs**: A link to the “Machine Learning Interpretability with Driverless AI” booklet.
- **Download MLI Logs**: Downloads a zip file of the logs that were generated during this interpretation.
- **Experiment**: Provides a link back to the experiment that generated this interpretation.
- **Scoring Pipeline**: Downloads the scoring pipeline for this interpretation.
- **Download Reason Codes**: Download a CSV file of LIME and/or Shapley reason codes.

### 15.6.1 Summary Page

The Summary page is the first page that opens when you view an interpretation. This page provides an overview of the interpretation, including the dataset and Driverless AI experiment (if available) that were used for the interpretation along with the feature space (original or transformed), target column, problem type, and k-Lime information. If the interpretation was created from a Driverless AI model, then a table with the Driverless AI model summary is also included along with the top variables for the model.

### 15.6.2 DAI Model Dropdown

This menu provides a Feature Importance plot and a Shapley plot (not supported for *RuleFit* and *TensorFlow* models) for transformed features.

**Feature Importance**

This plot shows the Driverless AI feature importance. Driverless AI feature importance is a measure of the contribution of an input variable to the overall predictions of the Driverless AI model. Global feature importance is calculated by
aggregating the improvement in splitting criterion caused by a single variable across all of the decision trees in the Driverless AI model.

Shapley Plot

Shapley explanations are a technique with credible theoretical support that presents consistent global and local variable contributions. Local numeric Shapley values are calculated by tracing single rows of data through a trained tree ensemble and aggregating the contribution of each input variable as the row of data moves through the trained ensemble.

For regression tasks, Shapley values sum to the prediction of the Driverless AI model. For classification problems, Shapley values sum to the prediction of the Driverless AI model before applying the link function. Global Shapley values are the average of the absolute Shapley values over every row of a data set.

Note: Shapley plots are not supported for RuleFit and TensorFlow models.


15.6.3 Surrogate Models Dropdown

The Surrogate Models dropdown includes KLIME/LIME-SUP and Decision Tree plots as well as a Random Forest submenu, which includes Global and Local Feature Importance plots for original features and a Partial Dependence plot.
Note: For multiclass classification experiments, only the Local Feature Importance plot is available in this dropdown.

K-LIME and LIME-SUP

The MLI screen includes a KLIME or LIME-SUP graph. A KLIME graph is available by default when you interpret a model from the experiment page. When you create a new interpretation, you can instead choose to use LIME-SUP as the LIME method. Note that these graphs are essentially the same, but the KLIME/LIME-SUP distinction provides insight into the LIME method that was used during model interpretation.

The K-LIME Technique

K-LIME is a variant of the LIME technique proposed by Ribeiro at al (2016). K-LIME generates global and local explanations that increase the transparency of the Driverless AI model, and allow model behavior to be validated and debugged by analyzing the provided plots, and comparing global and local explanations to one-another, to known standards, to domain knowledge, and to reasonable expectations.

K-LIME creates one global surrogate GLM on the entire training data and also creates numerous local surrogate GLMs on samples formed from \(k\)-means clusters in the training data. The features used for \(k\)-means are selected from the Random Forest surrogate model’s variable importance. The number of features used for \(k\)-means is the minimum of the top 25% of variables from the Random Forest surrogate model’s variable importance and the max number of variables that can be used for \(k\)-means, which is set by the user in the config.toml setting for \texttt{mli_max_number_cluster_vars}. (Note, if the number of features in the dataset are less than or equal to 6, then all features are used for \(k\)-means clustering.) The previous setting can be turned off to use all features for \(k\)-means by setting \texttt{use_all_columns_klime_kmeans} in the config.toml file to \texttt{true}. All penalized GLM surrogates are trained to model the predictions of the Driverless AI model. The number of clusters for local explanations is chosen by a grid search in which the \(R^2\) between the Driverless AI model predictions and all of the local K-LIME model predictions is maximized. The global and local linear model’s intercepts, coefficients, \(R^2\) values, accuracy, and predictions can all be used to debug and develop explanations for the Driverless AI model’s behavior.

The parameters of the global K-LIME model give an indication of overall linear feature importance and the overall average direction in which an input variable influences the Driverless AI model predictions. The global model is also used to generate explanations for very small clusters \((N < 20)\) where fitting a local linear model is inappropriate.

The in-cluster linear model parameters can be used to profile the local region, to give an average description of the important variables in the local region, and to understand the average direction in which an input variable affects the Driverless AI model predictions. For a point within a cluster, the sum of the local linear model intercept and the products of each coefficient with their respective input variable value are the K-LIME prediction. By disaggregating the K-LIME predictions into individual coefficient and input variable value products, the local linear impact of the variable can be determined. This product is sometimes referred to as a reason code and is used to create explanations for the Driverless AI model’s behavior.

In the following example, reason codes are created by evaluating and disaggregating a local linear model.

Given the row of input data with its corresponding Driverless AI and K-LIME predictions:

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>K-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

And the local linear model:

\[ y_{\text{K-LIME}} = 0.1 + 0.01 \times \text{debt_to_income_ratio} + 0.0005 \times \text{credit_score} + 0.0002 \times \text{savings_account_balance} \]

It can be seen that the local linear contributions for each variable are:
Each local contribution is positive and thus contributes positively to the Driverless AI model’s prediction of 0.85 for $H2OAI_{predicted\_default}$. By taking into consideration the value of each contribution, reason codes for the Driverless AI decision can be derived. $debt\_to\_income\_ratio$ and $credit\_score$ would be the two largest negative reason codes, followed by $savings\_acct\_balance$.

The local linear model intercept and the products of each coefficient and corresponding value sum to the K-LIME prediction. Moreover it can be seen that these linear explanations are reasonably representative of the nonlinear model’s behavior for this individual because the K-LIME predictions are within 5.5% of the Driverless AI model prediction. This information is encoded into English language rules which can be viewed by clicking the Explanations button.

Like all LIME explanations based on linear models, the local explanations are linear in nature and are offsets from the baseline prediction, or intercept, which represents the average of the penalized linear model residuals. Of course, linear approximations to complex non-linear response functions will not always create suitable explanations and users are urged to check the K-LIME plot, the local model $R^2$, and the accuracy of the K-LIME prediction to understand the validity of the K-LIME local explanations. When K-LIME accuracy for a given point or set of points is quite low, this can be an indication of extremely nonlinear behavior or the presence of strong or high-degree interactions in this local region of the Driverless AI response function. In cases where K-LIME linear models are not fitting the Driverless AI model well, nonlinear LOCO feature importance values may be a better explanatory tool for local model behavior. As K-LIME local explanations rely on the creation of $k$-means clusters, extremely wide input data or strong correlation between input variables may also degrade the quality of K-LIME local explanations.

The LIME-SUP Technique

LIME-SUP explains local regions of the trained Driverless AI model in terms of the original variables. Local regions are defined by each leaf node path of the decision tree surrogate model instead of simulated, perturbed observation samples - as in the original LIME. For each local region, a local GLM model is trained on the original inputs and the predictions of the Driverless AI model. Then the parameters of this local GLM can be used to generate approximate, local explanations of the Driverless AI model.

The Global Interpretable Model Explanation Plot

This plot shows Driverless AI model predictions and LIME model predictions in sorted order by the Driverless AI model predictions. This graph is interactive. Hover over the Model Prediction, LIME Model Prediction, or Actual Target radio buttons to magnify the selected predictions. Or click those radio buttons to disable the view in the graph. You can also hover over any point in the graph to view LIME reason codes for that value. By default, this plot shows information for the global LIME model, but you can change the plot view to show local results from a specific cluster. The LIME plot also provides a visual indication of the linearity of the Driverless AI model and the trustworthiness of the LIME explanations. The closer the local linear model approximates the Driverless AI model predictions, the more linear the Driverless AI model and the more accurate the explanation generated by the LIME local linear models.
The Decision Tree Surrogate Model Technique

The decision tree surrogate model increases the transparency of the Driverless AI model by displaying an approximate flow-chart of the complex Driverless AI model’s decision making process. The decision tree surrogate model also displays the most important variables in the Driverless AI model and the most important interactions in the Driverless AI model. The decision tree surrogate model can be used for visualizing, validating, and debugging the Driverless AI model by comparing the displayed decision-process, important variables, and important interactions to known standards, domain knowledge, and reasonable expectations.

A surrogate model is a data mining and engineering technique in which a generally simpler model is used to explain another, usually more complex, model or phenomenon. The decision tree surrogate is known to date back at least to 1996 (Craven and Shavlik). The decision tree surrogate model here is trained to predict the predictions of the more complex Driverless AI model using the of original model inputs. The trained surrogate model enables a heuristic understanding (i.e., not a mathematically precise understanding) of the mechanisms of the highly complex and nonlinear Driverless AI model.

The Decision Tree Plot

In the Decision Tree plot, the highlighted row shows the path to the highest probability leaf node and indicates the globally important variables and interactions that influence the Driverless AI model prediction for that row.
Random Forest Dropdown

The Random Forest dropdown provides a submenu that includes a Feature Importance plot, a Partial Dependence plot, and a LOCO plot. These plots are for original features rather than transformed features.

Feature Importance

Global Feature Importance vs Local Feature Importance

Global feature importance (yellow) is a measure of the contribution of an input variable to the overall predictions of the Driverless AI model. Global feature importance is calculated by aggregating the improvement in splitting criterion caused by a single variable across all of the decision trees in the Driverless AI model.

Local feature importance (grey) is a measure of the contribution of an input variable to a single prediction of the Driverless AI model. Local feature importance is calculated by removing the contribution of a variable from every decision tree in the Driverless AI model and measuring the difference between the prediction with and without the variable.

Both global and local variable importance are scaled so that the largest contributor has a value of 1.

Note: Engineered features are used for MLI when a time series experiment is built. This is because munged time series features are more useful features for MLI than raw time series features, as raw time series features are not IID (Independent and Identically Distributed).
LOCO

Local feature importance describes how the combination of the learned model rules or parameters and an individual row’s attributes affect a model’s prediction for that row while taking nonlinearity and interactions into effect. Local feature importance values reported here are based on a variant of the leave-one-covariate-out (LOCO) method (Lei et al, 2017).

In the LOCO-variant method, each local feature importance is found by re-scoring the trained Driverless AI model for each feature in the row of interest, while removing the contribution to the model prediction of splitting rules that contain that feature throughout the ensemble. The original prediction is then subtracted from this modified prediction to find the raw, signed importance for the feature. All local feature importance values for the row are then scaled between 0 and 1 for direct comparison with global feature importance values.

Given the row of input data with its corresponding Driverless AI and K-LIME predictions:

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>K-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>

Taking the Driverless AI model as \( F(X) \), LOCO-variant feature importance values are calculated as follows.

First, the modified predictions are calculated:

\[
F_{\text{debt_to_income_ratio}} = F(NA, 600, 1000) = 0.99 \\
F_{\text{credit_score}} = F(30, NA, 1000) = 0.73 \\
F_{\text{savings_acct_balance}} = F(30, 600, NA) = 0.82
\]

Second, the original prediction is subtracted from each modified prediction to generate the unscaled local feature importance values:

\[
\text{LOCO}_{\text{debt_to_income_ratio}} = F_{\text{debt_to_income_ratio}} - 0.85 = 0.99 - 0.85 = 0.14 \\
\text{LOCO}_{\text{credit_score}} = F_{\text{credit_score}} - 0.85 = 0.73 - 0.85 = -0.12 \\
\text{LOCO}_{\text{savings_acct_balance}} = F_{\text{savings_acct_balance}} - 0.85 = 0.82 - 0.85 = -0.03
\]

Finally LOCO values are scaled between 0 and 1 by dividing each value for the row by the maximum value for the row and taking the absolute magnitude of this quotient.
Scaled(LOCO\textsubscript{debt\_to\_income\_ratio}) = \text{Abs}(\text{LOCO} \text{ debt\_to\_income\_ratio}/0.14) = 1
Scaled(LOCO\textsubscript{credit\_score}) = \text{Abs}(\text{LOCO} \text{ credit\_score}/0.14) = 0.86
Scaled(LOCO\textsubscript{savings\_acct\_balance}) = \text{Abs}(\text{LOCO} \text{ savings\_acct\_balance}/0.14) = 0.21

One drawback to these LOCO-variant feature importance values is, unlike K-LIME, it is difficult to generate a mathematical error rate to indicate when LOCO values may be questionable.

**Partial Dependence and Individual Conditional Expectation (ICE)**

**The Partial Dependence Technique**

Partial dependence is a measure of the average model prediction with respect to an input variable. Partial dependence plots display how machine-learned response functions change based on the values of an input variable of interest, while taking nonlinearity into consideration and averaging out the effects of all other input variables. Partial dependence plots are well-known and described in the Elements of Statistical Learning (Hastie et al, 2001). Partial dependence plots enable increased transparency in Driverless AI models and the ability to validate and debug Driverless AI models by comparing a variable’s average predictions across its domain to known standards, domain knowledge, and reasonable expectations.

**The ICE Technique**

Individual conditional expectation (ICE) plots, a newer and less well-known adaptation of partial dependence plots, can be used to create more localized explanations for a single individual using the same basic ideas as partial dependence plots. ICE Plots were described by Goldstein et al (2015). ICE values are simply disaggregated partial dependence, but ICE is also a type of nonlinear sensitivity analysis in which the model predictions for a single row are measured while a variable of interest is varied over its domain. ICE plots enable a user to determine whether the model’s treatment of an individual row of data is outside one standard deviation from the average model behavior, whether the treatment of a specific row is valid in comparison to average model behavior, known standards, domain knowledge, and reasonable expectations, and how a model will behave in hypothetical situations where one variable in a selected row is varied across its domain.

Given the row of input data with its corresponding Driverless AI and K-LIME predictions:

<table>
<thead>
<tr>
<th>debt_to_income_ratio</th>
<th>credit_score</th>
<th>savings_acct_balance</th>
<th>observed_default</th>
<th>H2OAI_predicted_default</th>
<th>K-LIME_predicted_default</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>600</td>
<td>1000</td>
<td>1</td>
<td>0.85</td>
<td>0.9</td>
</tr>
</tbody>
</table>
Taking the Driverless AI model as $F(X)$, assuming credit scores vary from 500 to 800 in the training data, and that increments of 30 are used to plot the ICE curve, ICE is calculated as follows:

$$\text{ICE}_{\text{credit\_score},500} = F(30, 500, 1000)$$
$$\text{ICE}_{\text{credit\_score},530} = F(30, 530, 1000)$$
$$\text{ICE}_{\text{credit\_score},560} = F(30, 560, 1000)$$
$$\vdots$$
$$\text{ICE}_{\text{credit\_score},800} = F(30, 800, 1000)$$

The one-dimensional partial dependence plots displayed here do not take interactions into account. Large differences in partial dependence and ICE are an indication that strong variable interactions may be present. In this case partial dependence plots may be misleading because average model behavior may not accurately reflect local behavior.

**The Partial Dependence Plot**

Overlaying ICE plots onto partial dependence plots allow the comparison of the Driverless AI model’s treatment of certain examples or individuals to the model’s average predictions over the domain of an input variable of interest. This plot shows the partial dependence when a variable is selected and the ICE values when a specific row is selected. Users may select a point on the graph to see the specific value at that point. Partial dependence (yellow) portrays the average prediction behavior of the Driverless AI model across the domain of an input variable along with +/- 1 standard deviation bands. ICE (grey) displays the prediction behavior for an individual row of data when an input variable is toggled across its domain. Currently, partial dependence and ICE plots are only available for the top ten most important original input variables. Categorical variables with 20 or more unique values are never included in these plots.

**15.6.4 Dashboard Page**

The Model Interpretation Dashboard includes the following information:

- Global interpretable model explanation plot
- Feature importance (Global for original features; LOCO for interpretations with predictions and when interpreting on raw features)
- Decision tree surrogate model
- Partial dependence and individual conditional expectation plots

**Note:** The Dashboard is not available for multiclass classification experiments.
15.7 General Considerations

15.7.1 Machine Learning and Approximate Explanations

For years, common sense has deemed the complex, intricate formulas created by training machine learning algorithms to be uninterpretable. While great advances have been made in recent years to make these often nonlinear, non-monotonic, and non-continuous machine-learned response functions more understandable (Hall et al., 2017), it is likely that such functions will never be as directly or universally interpretable as more traditional linear models.

Why consider machine learning approaches for inferential purposes? In general, linear models focus on understanding and predicting average behavior, whereas machine-learned response functions can often make accurate, but more difficult to explain, predictions for subtler aspects of modeled phenomenon. In a sense, linear models create very exact interpretations for approximate models. The approach here seeks to make approximate explanations for very exact models. It is quite possible that an approximate explanation of an exact model may have as much, or more, value and meaning than the exact interpretations of an approximate model. Moreover, the use of machine learning techniques for inferential or predictive purposes does not preclude using linear models for interpretation (Ribeiro et al., 2016).

15.7.2 The Multiplicity of Good Models in Machine Learning

It is well understood that for the same set of input variables and prediction targets, complex machine learning algorithms can produce multiple accurate models with very similar, but not exactly the same, internal architectures (Breiman, 2001). This alone is an obstacle to interpretation, but when using these types of algorithms as interpretation tools or with interpretation tools it is important to remember that details of explanations will change across multiple accurate models.
15.7.3 Expectations for Consistency Between Explanatory Techniques

- The decision tree surrogate is a global, nonlinear description of the Driverless AI model behavior. Variables that appear in the tree should have a direct relationship with variables that appear in the global feature importance plot. For certain, more linear Driverless AI models, variables that appear in the decision tree surrogate model may also have large coefficients in the global K-LIME model.

- K-LIME explanations are linear, do not consider interactions, and represent offsets from the local linear model intercept. LOCO importance values are nonlinear, do consider interactions, and do not explicitly consider a linear intercept or offset. LIME explanations and LOCO importance values are not expected to have a direct relationship but can align roughly as both are measures of a variable’s local impact on a model’s predictions, especially in more linear regions of the Driverless AI model’s learned response function.

- ICE is a type of nonlinear sensitivity analysis which has a complex relationship to LOCO feature importance values. Comparing ICE to LOCO can only be done at the value of the selected variable that actually appears in the selected row of the training data. When comparing ICE to LOCO the total value of the prediction for the row, the value of the variable in the selected row, and the distance of the ICE value from the average prediction for the selected variable at the value in the selected row must all be considered.

- ICE curves that are outside the standard deviation of partial dependence would be expected to fall into less populated decision paths of the decision tree surrogate; ICE curves that lie within the standard deviation of partial dependence would be expected to belong to more common decision paths.

- Partial dependence takes into consideration nonlinear, but average, behavior of the complex Driverless AI model without considering interactions. Variables with consistently high partial dependence or partial dependence that swings widely across an input variable’s domain will likely also have high global importance values. Strong interactions between input variables can cause ICE values to diverge from partial dependence values.

15.8 Current Known Issues

- In some cases, after selecting a weight column, you may see that weight column used as an input to the decision tree and random forest surrogate models. This means the weight column could appear as an input in the surrogate decision tree, in the LOCO feature importance chart, or in other unexpected areas of the MLI dashboard.

- In some cases, after selecting a user-defined cluster column for K-LIME, you may see that cluster column used as an input to the decision tree and random forest surrogate models. This means the cluster column could appear as an input in the surrogate decision tree, in the LOCO feature importance chart, or in other unexpected areas of the MLI dashboard.

- Various menu behavior inconsistencies may exist, including an inability to unselect a target column and truncation of target column name.
CHAPTER
SIXTEEN

VIEWING EXPLANATIONS

Note: Not all explanatory functionality is available for multinomial classification scenarios.

Driverless AI provides easy-to-read explanations for a completed model. You can view these by clicking the Explanations button on the Model Interpretation page. Note that this button is only available for completed experiments. Click Close when you are done to return to the Model Interpretations page.

The UI allows you to view global, cluster-specific, and local reason codes. You can also export the explanations to CSV.

- **Global Reason Codes**: To view global reason codes, select the Global from the Cluster dropdown.

![Global Reason Codes](image)

With Global selected, click the Explanations button beside the Cluster dropdown.

- **Cluster Reason Codes**: To view reason codes for a specific cluster, select a cluster from the Cluster dropdown.
With a cluster selected, click the **Explanations** button.

**Local Reason Codes**: To view local reason codes, select a point on the graph or type a value in the Value field.

With a value selected, click the **Explanations** button.
### Actual and Predicted Values

<table>
<thead>
<tr>
<th>p(default payment next month) (Actual)</th>
<th>0.20</th>
</tr>
</thead>
<tbody>
<tr>
<td>k-LIME Prediction Value</td>
<td>0.27</td>
</tr>
<tr>
<td>k-LIME Prediction Accuracy</td>
<td>74.1%</td>
</tr>
</tbody>
</table>

### Local Reason Codes

#### k-LIME Local Attributes

<table>
<thead>
<tr>
<th>Variable</th>
<th>with value</th>
<th>Is associated with p(default payment next month)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Top Positive Local Attributes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39_NumToDxEDUCATIONPAY,PRAX,5D</td>
<td>0.4766917B8</td>
<td>increase of</td>
</tr>
<tr>
<td>34_WAGEPAY,PRAX,4D</td>
<td>0.16055322</td>
<td>increase of</td>
</tr>
<tr>
<td>28_PAY,5</td>
<td>2</td>
<td>increase of</td>
</tr>
<tr>
<td><strong>Skipped 32 additional attributes, click to view all...</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Top Negative Local Attributes</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>23_PAY,3</td>
<td>2</td>
<td>decrease of</td>
</tr>
<tr>
<td>10_DVSEXO</td>
<td>0.24495689</td>
<td>decrease of</td>
</tr>
<tr>
<td>3_DVTE,MARRAGE,O</td>
<td>0.20151251</td>
<td>decrease of</td>
</tr>
<tr>
<td><strong>Skipped 35 additional attributes, click to view all...</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>k-LIME Cluster Baseline Attribute</strong></td>
<td>0.37</td>
<td></td>
</tr>
</tbody>
</table>
Score on Another Dataset

After you generate a model, you can use that model to make predictions on another dataset.

1. Click the Experiments link in the top menu and select the experiment that you want to use.
2. On the Experiment page, click the Score on Another Dataset button.
3. Locate the new dataset that you want to score on. Note that this new dataset must include the same columns as the dataset used in selected experiment.
4. Click Select at the top of the screen. This immediately starts the scoring process.
5. Click the Download Predictions button after scoring is complete.
CHAPTER EIGHTEEN

TRANSFORM ANOTHER DATASET

When a training dataset is used in an experiment, Driverless AI transforms the data into an improved, feature engineered dataset. (Refer to *Driverless AI Transformations* for more information about the transformations that are provided in Driverless AI.) But what happens when new rows are added to your dataset? In this case, you can specify to transform the new dataset after adding it to Driverless AI, and the same transformations that Driverless AI applied to the original dataset will be applied to these new rows.

Follow these steps to transform another dataset. Note that this assumes the new dataset has been added to Driverless AI already.

**Note:** Transform Another Dataset is not available for Time Series experiments.

1. On the completed experiment page for the original dataset, click the Transform Another Dataset button.
2. Select the new training dataset that you want to transform. Note that this must have the same number columns as the original dataset.
3. In the Select drop down, specify a validation dataset to use with this dataset, or specify to split the training data. If you specify to split the data, then you also specify the split value (defaults to 25%) and the seed (defaults to 1234). **Note:** To ensure the transformed dataset respects the row order, choose a validation dataset instead of splitting the training data. Splitting the training data will result in a shuffling of the row order.
4. Optionally specify a test dataset. If specified, then the output also include the final test dataset for final scoring.
5. Click Launch Transformation.
The following datasets will be available for download upon successful completion:

- Training dataset (not for cross validation)
- Validation dataset for parameter tuning
- Test dataset for final scoring. This option is available if a test dataset was used.
THE PYTHON AND MOJO SCORING PIPELINES

Driverless AI provides a Python Scoring Pipeline for experiments and interpreted models and a MOJO (Java-based) Scoring Pipeline for experiments.

The Python Scoring Pipeline is implemented as a Python whl file. While this allows for a single process scoring engine, the scoring service is generally implemented as a client/server architecture and supports interfaces for TCP and HTTP.

The MOJO Scoring Pipeline provides a standalone scoring pipeline that converts experiments to MOJOs, which can be scored in real time.

Examples are included with each scoring package.

Note: These sections describe scoring pipelines and not deployments of scoring pipelines. For information on how to deploy a MOJO scoring pipeline, refer to the Deploying the MOJO Pipeline section.
Driverless AI provides a Python Scoring Pipeline, an MLI Standalone Scoring Pipeline, and a MOJO Scoring Pipeline. Consider the following when determining the scoring pipeline that you want to use.

- For all pipelines, the higher the accuracy, the slower the scoring.
- The Python Scoring Pipeline is slower but easier to use than the MOJO scoring pipeline.
- When running the Python Scoring Pipeline:
  - HTTP is easy and is supported by virtually any language. HTTP supports RESTful calls via curl, wget, or supported packages in various scripting languages.
  - TCP is a bit more complex, though faster. TCP also requires Thrift, which currently does not handle NAs.
- Use the MOJO Scoring Pipeline for a pure Java solution. This solution is flexible and is faster than the Python Scoring Pipeline, but it requires a bit more coding.
- The MLI Standalone Python Scoring Pipeline can be used to score interpreted models but only supports k-LIME reason codes.
  - For obtaining k-LIME reason codes from an MLI experiment, use the MLI Standalone Python Scoring Pipeline. k-LIME reason codes are available for all models.
  - For obtaining Shapley reason codes from an MLI experiment, use the DAI Standalone Python Scoring Pipeline. Shapley is only available for XGBoost and LightGBM models. Note that obtaining Shapley reason codes through the Python Scoring Pipeline can be time consuming.
As indicated earlier, a scoring pipeline is available after a successfully completed experiment. This package contains an exported model and Python 3.6 source code examples for productionizing models built using H2O Driverless AI. The files in this package allow you to transform and score on new data in a couple of different ways:

- From Python 3.6, you can import a scoring module, and then use the module to transform and score on new data.
- From other languages and platforms, you can use the TCP/HTTP scoring service bundled with this package to call into the scoring pipeline module through remote procedure calls (RPC).

### 21.1 Python Scoring Pipeline Files

The `scoring-pipeline` folder includes the following notable files:

- **example.py**: An example Python script demonstrating how to import and score new records.
- **run_example.sh**: Runs example.py (also sets up a virtualenv with prerequisite libraries).
- **tcp_server.py**: A standalone TCP server for hosting scoring services.
- **http_server.py**: A standalone HTTP server for hosting scoring services.
- **run_tcp_server.sh**: Runs TCP scoring service (runs tcp_server.py).
- **run_http_server.sh**: Runs HTTP scoring service (runs http_server.py).
- **example_client.py**: An example Python script demonstrating how to communicate with the scoring server.
- **run_tcp_client.sh**: Demonstrates how to communicate with the scoring service via TCP (runs example_client.py).
- **run_http_client.sh**: Demonstrates how to communicate with the scoring service via HTTP (using curl).

### 21.2 Quick Start

There are two methods for starting the Python Scoring Pipeline.

#### 21.2.1 Quick Start - Recommended Method

This is the recommended method for running the Python Scoring Pipeline. Use this method if:

- You have an air gapped environment with no access to the Internet.
Using Driverless AI, Release 1.6.2

• You are running Power.
• You want an easy quick start approach.

Prerequisites

• A valid Driverless AI license key.
• A completed Driverless AI experiment.
• Downloaded Python Scoring Pipeline.

Running the Python Scoring Pipeline - Recommended

1. Download the TAR SH version of Driverless AI from https://www.h2o.ai/download/ (for either Linux or IBM Power).
2. Use bash to execute the download. This creates a new `dai-nnn` folder.
3. Change directories into the new Driverless AI folder.
   ```bash
cd dai-nnn directory.
```
4. Run the following to install the Python Scoring Pipeline for your completed Driverless AI experiment:
   ```bash
./dai-env.sh pip install /path/to/your/scoring_experiment.whl
```
5. Run the following command to run the included scoring pipeline example:
   ```bash
DRIVERLESS_AI_LICENSE_KEY="pastekeyhere" SCORING_PIPELINE_INSTALL_DEPENDENCIES=0 ./dai-env.sh /path/to/your/run_example.sh
```

21.2.2 Quick Start - Alternative Method

Prerequisites

• The scoring module and scoring service are supported only on Linux with Python 3.6 and OpenBLAS.
• The scoring module and scoring service download additional packages at install time and require Internet access. Depending on your network environment, you might need to set up internet access via a proxy.
• Valid Driverless AI license. Driverless AI requires a license to be specified in order to run the Python Scoring Pipeline.
• Apache Thrift (to run the scoring service in TCP mode)
• Linux environment
• Python 3.6
• libopenblas-dev (required for H2O4GPU)
• OpenCL

Examples of how to install these prerequisites are below.

Installing Python 3.6

Installing Python 3.6 and OpenBLAS on Ubuntu 16.10+
Using Driverless AI, Release 1.6.2

Installing Python 3.6 and OpenBLAS on Ubuntu 16.04

```
sudo apt install python3.6 python3.6-dev python3-pip python3-dev 
   python-virtualenv python3-virtualenv libopenblas-dev
```

Installing Conda 3.6:

You can install Conda using either Anaconda or Miniconda. Refer to the links below for more information:

- Anaconda - https://docs.anaconda.com/anaconda/install.html

Installing OpenCL

Install OpenCL on RHEL

```
yum -y clean all
yum -y makecache
yum -y update
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/c/clinfo-2.1.17.02.09-1.˓→el7.x86_64.rpm
wget http://dl.fedoraproject.org/pub/epel/7/x86_64/Packages/o/ocl-icd-2.2.12-1.el7.˓→x86_64.rpm
rpm -if clinfo-2.1.17.02.09-1.el7.x86_64.rpm
rpm -if ocl-icd-2.2.12-1.el7.x86_64.rpm
clinfo
mkdir -p /etc/OpenCL/vendors && 
   echo "libnvidia-opencl.so.1" > /etc/OpenCL/vendors/nvidia.icd
```

Install OpenCL on Ubuntu

```
sudo apt install ocl-icd-libopencl1
mkdir -p /etc/OpenCL/vendors && 
   echo "libnvidia-opencl.so.1" > /etc/OpenCL/vendors/nvidia.icd
```

License Specification

Driverless AI requires a license to be specified in order to run the Python Scoring Pipeline. The license can be specified via an environment variable in Python:

```
# Set DRIVERLESS_AI_LICENSE_FILE, the path to the Driverless AI license file
%env DRIVERLESS_AI_LICENSE_FILE="/home/ubuntu/license/license.sig"

# Set DRIVERLESS_AI_LICENSE_KEY, the Driverless AI license key (Base64 encoded string)
%env DRIVERLESS_AI_LICENSE_KEY="oLqLZXMI0y..."
```

The examples that follow use DRIVERLESS_AI_LICENSE_FILE. Using DRIVERLESS_AI_LICENSE_KEY would be similar.

Installing the Thrift Compiler
Thrift is required to run the scoring service in TCP mode, but it is not required to run the scoring module. The following steps are available on the Thrift documentation site at: https://thrift.apache.org/docs/BuildingFromSource.

```
sudo apt-get install automake bison flex g++ git libevent-dev \
   libssl-dev libtool make pkg-config libboost-all-dev ant
wget https://github.com/apache/thrift/archive/0.10.0.tar.gz
tar -xvf 0.10.0.tar.gz
cd thrift-0.10.0
./bootstrap.sh
./configure
make
sudo make install
```

Run the following to refresh the runtime shared after installing Thrift:

```
sudo ldconfig /usr/local/lib
```

**Running the Python Scoring Pipeline - Alternative Method**

1. On the completed Experiment page, click on the Download Python Scoring Pipeline button to download the scorer.zip file for this experiment onto your local machine.

2. Unzip the scoring pipeline.

After the pipeline is downloaded and unzipped, you will be able to run the scoring module and the scoring service.

**Score from a Python Program**

If you intend to score from a Python program, run the scoring module example. (Requires Linux and Python 3.6.)
Score Using a Web Service

If you intend to score using a web service, run the HTTP scoring server example. (Requires Linux x86_64 and Python 3.6.)

```bash
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_example.sh
bash run_http_server.sh
bash run_http_client.sh
```

Score Using a Thrift Service

If you intend to score using a Thrift service, run the TCP scoring server example. (Requires Linux x86_64, Python 3.6 and Thrift.)

```bash
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_tcp_server.sh
bash run_tcp_client.sh
```

Note: By default, the `run_*`.sh scripts mentioned above create a virtual environment using virtualenv and pip, within which the Python code is executed. The scripts can also leverage Conda (Anaconda/Mininconda) to create Conda virtual environment and install required package dependencies. The package manager to use is provided as an argument to the script.

```bash
# to use conda package manager
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_example.sh --pm conda

# to use pip package manager
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_example.sh --pm pip
```

If you experience errors while running any of the above scripts, please check to make sure your system has a properly installed and configured Python 3.6 installation. Refer to the Troubleshooting Python Environment Issues section that follows to see how to set up and test the scoring module using a cleanroom Ubuntu 16.04 virtual machine.

## 21.3 The Python Scoring Module

The scoring module is a Python module bundled into a standalone wheel file (name scoring_*). All the prerequisites for the scoring module to work correctly are listed in the requirements.txt file. To use the scoring module, all you have to do is create a Python virtualenv, install the prerequisites, and then import and use the scoring module as follows:

```python
# See 'example.py' for complete example.
from scoring_487931_20170921174120_b4066 import Scorer
scorer = Scorer()  # Create instance.
score = scorer.score([  # Call score()
    7.416,  # sepal_len
    3.562,  # sepal_wid
    1.049,  # petal_len
    2.388,  # petal_wid
])
```

The scorer instance provides the following methods (and more):

### 21.3. The Python Scoring Module
Using Driverless AI, Release 1.6.2

- score(list): Score one row (list of values).
- score_batch(df): Score a Pandas dataframe.
- fit_transform_batch(df): Transform a Pandas dataframe.
- get_target_labels(): Get target column labels (for classification problems).

The process of importing and using the scoring module is demonstrated by the bash script `run_example.sh`, which effectively performs the following steps:

```bash
# See 'run_example.sh' for complete example.
virtualenv --p python3.6 env
source env/bin/activate
pip install -r requirements.txt
export DRIVERLESS.AI_LICENSE_FILE="/path/to/license.sig"
python example.py
```

21.4 The Scoring Service

The scoring service hosts the scoring module as an HTTP or TCP service. Doing this exposes all the functions of the scoring module through remote procedure calls (RPC). In effect, this mechanism allows you to invoke scoring functions from languages other than Python on the same computer or from another computer on a shared network or on the Internet.

The scoring service can be started in two ways:

- In TCP mode, the scoring service provides high-performance RPC calls via Apache Thrift (https://thrift.apache.org/) using a binary wire protocol.
- In HTTP mode, the scoring service provides JSON-RPC 2.0 calls served by Tornado (http://www.tornadoweb.org).

Scoring operations can be performed on individual rows (row-by-row) or in batch mode (multiple rows at a time).

21.4.1 Scoring Service - TCP Mode (Thrift)

The TCP mode allows you to use the scoring service from any language supported by Thrift, including C, C++, C#, Cocoa, D, Dart, Delphi, Go, Haxe, Java, Node.js, Lua, perl, PHP, Python, Ruby and Smalltalk.

To start the scoring service in TCP mode, you will need to generate the Thrift bindings once, then run the server:

```bash
# See 'run_tcp_server.sh' for complete example.
thrift --gen py scoring.thrift
tcp_server.py --port=9090
```

Note that the Thrift compiler is only required at build-time. It is not a run time dependency, i.e. once the scoring services are built and tested, you do not need to repeat this installation process on the machines where the scoring services are intended to be deployed.

To call the scoring service, simply generate the Thrift bindings for your language of choice, then make RPC calls via TCP sockets using Thrift’s buffered transport in conjunction with its binary protocol.

```bash
# See 'run_tcp_client.sh' for complete example.
thrift --gen py scoring.thrift
# See 'example_client.py' for complete example.
```

(continues on next page)
socket = TSocket.TSocket('localhost', 9090)
transport = TTransport.TBufferedTransport(socket)
protocol = TBinaryProtocol.TBinaryProtocol(transport)
client = ScoringService.Client(protocol)
transport.open()
row = Row()
row.sepalLen = 7.416  # sepal_len
row.sepalWid = 3.562  # sepal_wid
row.petalLen = 1.049  # petal_len
row.petalWid = 2.388  # petal_wid
scores = client.score(row)
transport.close()

You can reproduce the exact same result from other languages, e.g. Java:

```
thrift --gen java scoring.thrift

// Dependencies:
// commons-codec-1.9.jar
// commons-logging-1.2.jar
// httpclient-4.4.1.jar
// httpcore-4.4.1.jar
// libthrift-0.10.0.jar
// slf4j-api-1.7.12.jar

import ai.h2o.scoring.Row;
import ai.h2o.scoring.ScoringService;
import org.apache.thrift.TException;
import org.apache.thrift.protocol.TBinaryProtocol;
import org.apache.thrift.transport.TSocket;
import org.apache.thrift.transport.TTransport;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        try {
            TTransport transport = new TSocket("localhost", 9090);
            transport.open();

            ScoringService.Client client = new ScoringService.Client(
                    new TBinaryProtocol(transport));

            Row row = new Row(7.642, 3.436, 6.721, 1.020);
            List<Double> scores = client.score(row);
            System.out.println(scores);
            transport.close();
        } catch (TException ex) {
            ex.printStackTrace();
         }
    }
}
```
21.4.2 Scoring Service - HTTP Mode (JSON-RPC 2.0)

The HTTP mode allows you to use the scoring service using plaintext JSON-RPC calls. This is usually less performant compared to Thrift, but has the advantage of being usable from any HTTP client library in your language of choice, without any dependency on Thrift.

For JSON-RPC documentation, see http://www.jsonrpc.org/specification.

To start the scoring service in HTTP mode:

```bash
# See 'run_http_server.sh' for complete example.
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
python http_server.py --port=9090
```

To invoke scoring methods, compose a JSON-RPC message and make a HTTP POST request to http://host:port/rpc as follows:

```bash
# See 'run_http_client.sh' for complete example.
curl http://localhost:9090/rpc \
   --header "Content-Type: application/json" \ 
   --data @- <<EOF
   { 
     "id": 1, 
     "method": "score", 
     "params": { 
       "row": [ 7.486, 3.277, 4.755, 2.354 ] 
     } 
   }
EOF
```

Similarly, you can use any HTTP client library to reproduce the above result. For example, from Python, you can use the requests module as follows:

```python
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()]['result'])
```

21.5 Python Scoring Pipeline FAQ

Why am I getting a “TensorFlow is disabled” message when I run the Python Scoring Pipeline?

If you ran an experiment when TensorFlow was enabled and then attempt to run the Python Scoring Pipeline, you may receive a message similar to the following:

```
TensorFlow is disabled. To enable, export DRIVERLESS_AI_ENABLE_TENSORFLOW=1 or set
   enable_tensorflow=true in config.toml.
```

To successfully run the Python Scoring Pipeline, you must enable the DRIVERLESS_AI_ENABLE_TENSORFLOW flag. For example:

```bash
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
DRIVERLESS_AI_ENABLE_TENSORFLOW=1 bash run_example.sh
```
21.6 Troubleshooting Python Environment Issues

The following instructions describe how to set up a cleanroom Ubuntu 16.04 virtual machine to test that this scoring pipeline works correctly.

Prerequisites:

- Install Virtualbox: `sudo apt-get install virtualbox`
- Install Vagrant: https://www.vagrantup.com/downloads.html

1. Create configuration files for Vagrant.
   - bootstrap.sh: contains commands to set up Python 3.6 and OpenBLAS.
   - Vagrantfile: contains virtual machine configuration instructions for Vagrant and VirtualBox.

```
----- bootstrap.sh -----
#!/usr/bin/env bash

sudo apt-get -y update
sudo apt-get -y install apt-utils build-essential python-software-properties
sudo add-apt-repository -y ppa:deadsnakes/ppa
sudo apt-get update -yqq
sudo apt-get install -y python3.6 python3.6-dev python3-pip python3-dev

# end of bootstrap.sh

----- Vagrantfile -----

Vagrant.configure(2) do |config|
  config.vm.box = "ubuntu/xenial64"
  config.vm.provision :shell, path: "bootstrap.sh", privileged: false
  config.vm.hostname = "h2o"
  config.vm.provider "virtualbox" do |vb|
    vb.memory = "4096"
  end
end

# end of Vagrantfile
```

2. Launch the VM and SSH into it. Note that we’re also placing the scoring pipeline in the same directory so that we can access it later inside the VM.

```
cp /path/to/scorer.zip .
vagrant up
vagrant ssh
```

3. Test the scoring pipeline inside the virtual machine.

```
cp /vagrant/scorer.zip .
unzip scorer.zip
cd scoring-pipeline/
```

(continues on next page)
export DRIVERLESS_AI_LICENSE_FILE="/path/to/license.sig"
bash run_example.sh

At this point, you should see scores printed out on the terminal. If not, contact us at support@h2o.ai.
CHAPTER TWENTYTWO

DRIVERLESS AI MLI STANDALONE PYTHON SCORING PACKAGE

This package contains an exported model and Python 3.6 source code examples for productionizing models built using H2O Driverless AI Machine Learning Interpretability (MLI) tool. This is only available for interpreted models and can be downloaded by clicking the Scoring Pipeline button the Interpreted Models page.

The files in this package allow you to obtain reason codes for a given row of data a couple of different ways:

- From Python 3.6, you can import a scoring module, and then use the module to transform and score on new data.
- From other languages and platforms, you can use the TCP/HTTP scoring service bundled with this package to call into the scoring pipeline module through remote procedure calls (RPC).

22.1 MLI Python Scoring Package Files

The `scoring-pipeline-mli` folder includes the following notable files:

- `example.py`: An example Python script demonstrating how to import and interpret new records.
- `run_example.sh`: Runs example.py (This also sets up a virtualenv with prerequisite libraries.)
- `run_example_shapley.sh`: Runs example_shapley.py. This compares K-LIME and Driverless AI Shapley reason codes.
- `tcp_server.py`: A standalone TCP server for hosting MLI services.
- `http_server.py`: A standalone HTTP server for hosting MLI services.
- `run_tcp_server.sh`: Runs the TCP scoring service (specifically, tcp_server.py).
- `run_http_server.sh`: Runs HTTP scoring service (runs http_server.py).
- `example_client.py`: An example Python script demonstrating how to communicate with the MLI server.
- `example_shapley.py`: An example Python script demonstrating how to compare K-LIME and Driverless AI Shapley reason codes.
- `run_tcp_client.sh`: Demonstrates how to communicate with the MLI service via TCP (runs example_client.py).
- `run_http_client.sh`: Demonstrates how to communicate with the MLI service via HTTP (using curl).

22.2 Quick Start

There are two methods for starting the MLI Standalone Scoring Pipeline.
22.2.1 Quick Start - Recommended Method

This is the recommended method for running the MLI Scoring Pipeline. Use this method if:

- You have an air gapped environment with no access to the Internet.
- You are running Power.
- You want an easy quick start approach.

Prerequisites

- A valid Driverless AI license key.
- A completed Driverless AI experiment.
- Downloaded MLI Scoring Pipeline.

Running the MLI Scoring Pipeline - Recommended

1. Download the TAR SH version of Driverless AI from https://www.h2o.ai/download/ (for either Linux or IBM Power).
2. Use bash to execute the download. This creates a new `dai-nnn` folder.
3. Change directories into the new Driverless AI folder.
   
   ```
   cd dai-nnn directory.
   ```
4. Run the following to install the Python Scoring Pipeline for your completed Driverless AI experiment:
   
   ```
   ./dai-env.sh pip install /path/to/your/scoring_experiment.whl
   ```
5. Run the following command to run the included scoring pipeline example:
   
   ```
   DRIVERLESS_AI_LICENSE_KEY="pastekeyhere" SCORING_PIPELINE_INSTALL_DEPENDENCIES=0 ./dai-env.sh /path/to/your/run_example.sh
   ```

22.2.2 Quick Start - Alternative Method

This section describes an alternative method for running the MLI Standalone Scoring Pipeline. This version requires Internet access. It is also not supported on Power machines.

Prerequisites

- Valid Driverless AI license.
- The scoring module and scoring service are supported only on Linux with Python 3.6 and OpenBLAS.
- The scoring module and scoring service download additional packages at install time and require internet access. Depending on your network environment, you might need to set up internet access via a proxy.
- Apache Thrift (to run the scoring service in TCP mode)
Examples of how to install these prerequisites are below.

**Installing Python 3.6**

Installing Python3.6 on Ubuntu 16.10+:

```bash
sudo apt install python3.6 python3.6-dev python3-pip python3-dev \
python-virtualenv python3-virtualenv
```

Installing Python3.6 on Ubuntu 16.04:

```bash
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.6 python3.6-dev python3-pip python3-dev \
python-virtualenv python3-virtualenv
```

Installing Conda 3.6:

You can install Conda using either Anaconda or Miniconda. Refer to the links below for more information:

- Anaconda - https://docs.anaconda.com/anaconda/install.html

**Installing the Thrift Compiler**

Refer to Thrift documentation at https://thrift.apache.org/docs/BuildingFromSource for more information.

```bash
sudo apt-get install automake bison flex g++ git libevent-dev \
libssl-dev libtool make pkg-config libboost-all-dev ant
wget https://github.com/apache/thrift/archive/0.10.0.tar.gz
 tar -xvf 0.10.0.tar.gz
cd thrift-0.10.0
./bootstrap.sh
./configure
make
sudo make install
```

Run the following to refresh the runtime shared after installing Thrift.

```bash
sudo ldconfig /usr/local/lib
```

**Running the MLI Scoring Pipeline - Alternative Method**

1. On the MLI page, click the **Scoring Pipeline** button.

2. Unzip the scoring pipeline, and run the following examples in the `scoring-pipeline-mli` folder.
   - Run the scoring module example. (This requires Linux and Python 3.6.)
     ```bash
     bash run_example.sh
     ```
   - Run the TCP scoring server example. Use two terminal windows. (This requires Linux, Python 3.6 and Thrift.)
Run the HTTP scoring server example. Use two terminal windows. (This requires Linux, Python 3.6 and Thrift.)

Note: By default, the run_*.sh scripts mentioned above create a virtual environment using virtualenv and pip, within which the Python code is executed. The scripts can also leverage Conda (Anaconda/Miniconda) to create Conda virtual environment and install required package dependencies. The package manager to use is provided as an argument to the script.

```
# to use conda package manager
bash run_example.sh --pm conda
# to use pip package manager
bash run_example.sh --pm pip
```

### 22.3 MLI Python Scoring Module

The MLI scoring module is a Python module bundled into a standalone wheel file (name scoring_*.whl). All the prerequisites for the scoring module to work correctly are listed in the ‘requirements.txt’ file. To use the scoring module, all you have to do is create a Python virtualenv, install the prerequisites, and then import and use the scoring module as follows:

```
----- See 'example.py' for complete example. ----- 
from scoring_487931_20170921174120_b4066 import Scorer
scorer = KLimeScorer()  # Create instance.
score = scorer.score_reason_codes([  # Call score_reason_codes()
    7.416,  # sepal_len
    3.562,  # sepal_wid
    1.049,  # petal_len
    2.388,  # petal_wid
])
```

The scorer instance provides the following methods:

- `score_reason_codes(list)`: Get K-LIME reason codes for one row (list of values).
- `score_reason_codes_batch(dataframe)`: Takes and outputs a Pandas Dataframe
- `get_column_names()`: Get the input column names
- `get_reason_code_column_names()`: Get the output column names

The process of importing and using the scoring module is demonstrated by the bash script `run_example.sh`, which effectively performs the following steps:

```
----- See 'run_example.sh' for complete example. ----- 
virtualenv -p python3.6 env
source env/bin/activate
pip install -r requirements.txt
python example.py
```
22.4 K-LIME vs Shapley Reason Codes

There are times when the K-LIME model score is not close to the Driverless AI model score. In this case it may be better to use reason codes using the Shapley method on the Driverless AI model. Please note: the reason codes from Shapley will be in the transformed feature space.

To see an example of using both K-LIME and Driverless AI Shapley reason codes in the same Python session, run:

```
bash run_example_shapley.sh
```

For this batch script to succeed, MLI must be run on a Driverless AI model. If you have run MLI in standalone (external model) mode, there will not be a Driverless AI scoring pipeline.

If MLI was run with transformed features, the Shapley example scripts will not be exported. You can generate exact reason codes directly from the Driverless AI model scoring pipeline.

22.5 MLI Scoring Service Overview

The MLI scoring service hosts the scoring module as a HTTP or TCP service. Doing this exposes all the functions of the scoring module through remote procedure calls (RPC).

In effect, this mechanism allows you to invoke scoring functions from languages other than Python on the same computer, or from another computer on a shared network or the internet.

The scoring service can be started in two ways:

- In TCP mode, the scoring service provides high-performance RPC calls via Apache Thrift (https://thrift.apache.org/) using a binary wire protocol.
- In HTTP mode, the scoring service provides JSON-RPC 2.0 calls served by Tornado (http://www.tornadoweb.org).

Scoring operations can be performed on individual rows (row-by-row) using `score` or in batch mode (multiple rows at a time) using `score_batch`. Both functions allow you to specify `pred_contribs=[True|False]` to get MLI predictions (KLime/Shapley) on a new dataset. See the `example_shapley.py` file for more information.

22.5.1 MLI Scoring Service - TCP Mode (Thrift)

The TCP mode allows you to use the scoring service from any language supported by Thrift, including C, C++, C#, Cocoa, D, Dart, Delphi, Go, Haxe, Java, Node.js, Lua, perl, PHP, Python, Ruby and Smalltalk.

To start the scoring service in TCP mode, you will need to generate the Thrift bindings once, then run the server:

```
----- See 'run_tcp_server.sh' for complete example. -----  
thrift --gen py scoring.thrift
python tcp_server.py --port=9090
```

Note that the Thrift compiler is only required at build-time. It is not a run time dependency, i.e. once the scoring services are built and tested, you do not need to repeat this installation process on the machines where the scoring services are intended to be deployed.

To call the scoring service, simply generate the Thrift bindings for your language of choice, then make RPC calls via TCP sockets using Thrift’s buffered transport in conjunction with its binary protocol.
Using Driverless AI, Release 1.6.2

----- See 'run_tcp_client.sh' for complete example. -----  
thrift --gen py scoring.thrift  

----- See 'example_client.py' for complete example. -----  
socket = TSocket.TSocket('localhost', 9090)  
transport = TTransport.TBufferedTransport(socket)  
protocol = TBinaryProtocol.TBinaryProtocol(transport)  
client = ScoringService.Client(protocol)  
transport.open()  
row = Row()  
row.sepalLen = 7.416 # sepal_len  
row.sepalWid = 3.562 # sepal_wid  
row.petalLen = 1.049 # petal_len  
row.petalWid = 2.388 # petal_wid  
scores = client.score_reason_codes(row)  
transport.close()  

You can reproduce the exact same result from other languages, e.g. Java:

thrift --gen java scoring.thrift  

// Dependencies:  
// commons-codec-1.9.jar  
// commons-logging-1.2.jar  
// httpclient-4.4.1.jar  
// httpcore-4.4.1.jar  
// libthrift-0.10.0.jar  
// slf4j-api-1.7.12.jar  

import ai.h2o.scoring.Row;  
import ai.h2o.scoring.ScoringService;  
import org.apache.thrift.TException;  
import org.apache.thrift.protocol.TBinaryProtocol;  
import org.apache.thrift.transport.TSocket;  
import org.apache.thrift.transport.TTransport;  
import java.util.List;  

public class Main {  
  public static void main(String[] args) {  
    try {  
      TTransport transport = new TSocket("localhost", 9090);  
      transport.open();  

      ScoringService.Client client = new ScoringService.Client(  
        new TBinaryProtocol(transport));  

      Row row = new Row(7.642, 3.436, 6.721, 1.020);  
      List<Double> scores = client.score_reason_codes(row);  
      System.out.println(scores);  

      transport.close();  
    } catch (TException ex) {  
      ex.printStackTrace();  
    }  
  }  
}

300 Chapter 22. Driverless AI MLI Standalone Python Scoring Package
22.5.2 Scoring Service - HTTP Mode (JSON-RPC 2.0)

The HTTP mode allows you to use the scoring service using plaintext JSON-RPC calls. This is usually less performant compared to Thrift, but has the advantage of being usable from any HTTP client library in your language of choice, without any dependency on Thrift.

For JSON-RPC documentation, see [http://www.jsonrpc.org/specification](http://www.jsonrpc.org/specification).

To start the scoring service in HTTP mode:

```
----- See 'run_http_server.sh' for complete example. -----  
python http_server.py --port=9090
```

To invoke scoring methods, compose a JSON-RPC message and make a HTTP POST request to `http://host:port/rpc` as follows:

```
----- See 'run_http_client.sh' for complete example. -----  
curl http://localhost:9090/rpc  
    --header "Content-Type: application/json"  
    --data @- <<EOF
    {
      "id": 1,
      "method": "score_reason_codes",
      "params": {
        "row": [ 7.486, 3.277, 4.755, 2.354 ]
      }
    }

EOF
```

Similarly, you can use any HTTP client library to reproduce the above result. For example, from Python, you can use the `requests` module as follows:

```python
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score_reason_codes', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()["result"])
```
For completed experiments, Driverless AI converts models to MOJOs (Model Objects, Optimized). A MOJO is a scoring engine that can be deployed in any Java environment for scoring in real time.

Keep in mind that, similar to H2O-3, MOJOs are tied to experiments. Experiments and MOJOs are not automatically upgraded when Driverless AI is upgraded.

Note: MOJOs are currently not available for TensorFlow, RuleFit, or FTRL models.

### 23.1 Prerequisites

The following are required in order to run the MOJO scoring pipeline.

- Java 8 runtime
- Valid Driverless AI license. You can download the `license.sig` file from the machine hosting Driverless AI (usually in the `license` folder). Copy the license file into the downloaded `mojo-pipeline` folder.
- `mojo2-runtime.jar` file. This is available from the top navigation menu in the Driverless AI UI and in the downloaded `mojo-pipeline.zip` file for an experiment.

### 23.1.1 License Specification

Driverless AI requires a license to be specified in order to run the MOJO Scoring Pipeline. The license can be specified in one of the following ways:

- **Via an environment variable:**
  - `DRIVERLESS_AI_LICENSE_FILE`: Path to the Driverless AI license file, or
  - `DRIVERLESS_AI_LICENSE_KEY`: The Driverless AI license key (Base64 encoded string)

- **Via a system property of JVM (`-D` option):**
  - `ai.h2o.mojos.runtime.license.file`: Path to the Driverless AI license file, or
  - `ai.h2o.mojos.runtime.license.key`: The Driverless AI license key (Base64 encoded string)

- **Via an application classpath:**
  - The license is loaded from a resource called `/license.sig`.
  - The default resource name can be changed via the JVM system property `ai.h2o.mojos.runtime.license.filename`.

For example:
23.2 Enabling the MOJO Scoring Pipeline

The MOJO Scoring Pipeline is disabled by default. As a result, a MOJO will have to be built for each desired experiment by clicking on the Build MOJO Scoring Pipeline button:

![Build MOJO Scoring Pipeline button]

To enable MOJO Scoring Pipelines for each experiment, stop Driverless AI, then restart using the DRIVERLESS_AI_MAKE_MOJO_SCORING_PIPELINE=1 flag. (Refer to Using the config.toml File section for more information.) For example:

```bash
nvidia-docker run \
--add-host name.node:172.16.2.186 \
-e DRIVERLESS_AI_MAKE_MOJO_SCORING_PIPELINE=1 \
-p 12345:12345 \
--pid=host \
--init \
--rm \
-v /tmp/dtmp/:/tmp \
-v /tmp/dlog/:/log \
-u $(id -u):$(id -g) \
opsh2oai/h2oai-runtime
```

Or you can change the value of make_mojo_scoring_pipeline to true in the config.toml file and specify that file when restarting Driverless AI.

23.3 MOJO Scoring Pipeline Files

The mojo-pipeline folder includes the following files:

- **run_example.sh**: An bash script to score a sample test set.
- **pipeline.mojo**: Standalone scoring pipeline in MOJO format.
- **mojo2-runtime.jar**: MOJO Java runtime.
- **example.csv**: Sample test set (synthetic, of the correct format).

23.4 Quickstart

Before running the quickstart examples, be sure that the MOJO scoring pipeline is already downloaded and unzipped:

1. On the completed Experiment page, click on the Download Scoring Pipeline button to download the scorer.zip file for this experiment onto your local machine.
**Note:** This button is **Build MOJO Scoring Pipeline** if the MOJO Scoring Pipeline is disabled.

2. To score all rows in the sample test set (example.csv) with the MOJO pipeline (pipeline.mojo) and license stored in the environment variable DRIVERLESS_AI_LICENSE_KEY:

   ```bash
   bash run_example.sh
   ```

3. To score a specific test set (example.csv) with MOJO pipeline (pipeline.mojo) and the license file (license.sig):

   ```bash
   bash run_example.sh pipeline.mojo example.csv license.sig
   ```

4. To run the Java application for data transformation directly:

   ```bash
   java -Dai.h2o.mojos.runtime.license.file=license.sig -cp mojo2-runtime.jar ai.h2o.mojos.ExecuteMojo pipeline.mojo example.csv
   ```

   **Note:** For very large models, it may be necessary to increase the memory limit when running the Java application for data transformation. This can be done by specifying `-Xmx25g` when running the above command.

### 23.5 Compile and Run the MOJO from Java

1. Open a **new** terminal window and change directories to the experiment folder:

   ```bash
   cd experiment
   ```

2. Create your main program in the experiment folder by creating a new file called **Main.java** (for example, using `vim Main.java`). Include the following contents.
import java.io.IOException;
import ai.h2o.mojos.runtime.MojoPipeline;
import ai.h2o.mojos.runtime.frame.MojoFrame;
import ai.h2o.mojos.runtime.frame.MojoFrameBuilder;
import ai.h2o.mojos.runtime.frame.MojoRowBuilder;
import ai.h2o.mojos.runtime.utils.SimpleCSV;
import ai.h2o.mojos.runtime.lic.LicenseException;

public class Main {

    public static void main(String[] args) throws IOException, LicenseException {
        // Load model and csv
        MojoPipeline model = MojoPipeline.loadFrom("pipeline.mojo");

        // Get and fill the input columns
        MojoFrameBuilder frameBuilder = model.getInputFrameBuilder();
        MojoRowBuilder rowBuilder = frameBuilder.getMojoRowBuilder();
        rowBuilder.setValue("AGE", "68");
        rowBuilder.setValue("RACE", "2");
        rowBuilder.setValue("DCAPS", "2");
        rowBuilder.setValue("VOL", "0");
        rowBuilder.setValue("GLEASON", "6");
        frameBuilder.addRow(rowBuilder);

        // Create a frame which can be transformed by MOJO pipeline
        MojoFrame iframe = frameBuilder.toMojoFrame();

        // Transform input frame by MOJO pipeline
        MojoFrame oframe = model.transform(iframe);

        // `MojoFrame.debug()` can be used to view the contents of a Frame
        // oframe.debug();

        // Output prediction as CSV
        SimpleCSV outCsv = SimpleCSV.read(oframe);
        outCsv.write(System.out);
    }
}

3. Compile the source code:
```
javac -cp mojo2-runtime.jar -J-Xms2g -J-XX:MaxPermSize=128m Main.java
```

4. Run the MOJO example:
```
# Linux and OS X users
java -Dai.h2o.mojos.runtime.license.file=license.sig -cp .:mojo2-runtime.jar -Main
# Windows users
java -Dai.h2o.mojos.runtime.license.file=license.sig -cp .;mojo2-runtime.jar -Main
```

5. The following output is displayed:
```
CAPSULE.True
0.5442205910902282
```
23.6 Using the MOJO Scoring Pipeline with Spark/Sparkling Water

Note: The Driverless AI 1.5 release will be the last release with TOML-based MOJO2. Releases after 1.5 will include protobuf-based MOJO2.

MOJO scoring pipeline artifacts can be used in Spark to deploy predictions in parallel using the Sparkling Water API. This section shows how to load and run predictions on the MOJO scoring pipeline in Spark using Scala and the Python API.

In the event that you upgrade H2O Driverless AI, we have a good news! Sparkling Water is backwards compatible with MOJO versions produced by older Driverless AI versions.

23.6.1 Requirements

- You must have a Spark cluster with the Sparkling Water JAR file passed to Spark.
- To run with PySparkling, you must have the PySparkling zip file.

The H2OContext does not have to be created if you only want to run predictions on MOJOs using Spark. This is because they are written to be independent of the H2O run-time.

23.6.2 Preparing Your Environment

Both PySparkling and Sparkling Water need to be started with some extra configurations in order to enable the MOJO scoring pipeline. Examples are provided below. Specifically, you must pass the path of the H2O Driverless AI license to the Spark --jars argument. Additionally, you need to add to the same --jars configuration path to the MOJO scoring pipeline implementation JAR file mojo2-runtime.jar. This file is proprietary and is not part of the resulting Sparkling Water assembly JAR file.

Note: In Local Spark mode, please use --driver-class-path to specify path to the license file and the MOJO Pipeline JAR file.

23.6.3 PySparkling

First, start PySpark with all the required dependencies. The following command passes the license file and the MOJO scoring pipeline implementation library to the --jars argument and also specifies the path to the PySparkling Python library.

```bash
./bin/pyspark --jars license.sig,mojo2-runtime.jar --py-files pysparkling.zip
```

or, you can download official Sparkling Water distribution from H2O Download page. Please follow steps on the Sparkling Water download page. Once you are in the Sparkling Water directory, you can call:

```bash
./bin/pysparkling --jars license.sig,mojo2-runtime.jar
```

At this point, you should have available a PySpark interactive terminal where you can try out predictions. If you would like to productionalize the scoring process, you can use the same configuration, except instead of using ./bin/pyspark, you would use ./bin/spark-submit to submit your job to a cluster.

```python
# First, specify the dependency
from pysparkling.ml import H2OMOJOPipelineModel
```
Using Driverless AI, Release 1.6.2

```python
# Load the pipeline
mojo = H2OMOJOPipelineModel.create_from_mojo("file:///path/to/the/pipeline.mojo")

# This option ensures that the output columns are named properly. If you want to use
# old behavior
# when all output columns were stored inside an array, don’t specify this
# configuration option,
# or set it to False. We however strongly encourage users to set this to True as
# below.
mojo.set_named_mojo_output_columns(True)
```

```python
# Load the data as Spark's Data Frame
data_frame = spark.read.csv("file:///path/to/the/data.csv", header=True)
```

```python
# Run the predictions. The predictions contain all the original columns plus the
# predictions
# added as new columns
predictions = mojo.predict(data_frame)

# You can easily get the predictions for a desired column using the helper function as
predictions.select(mojo.select_prediction_udf("AGE")).collect()
```

### 23.6.4 Sparkling Water

First start Spark with all the required dependencies. The following command passes the license file and the MOJO
scoring pipeline implementation library mojo2-runtime.jar to the --jars argument and also specifies the path to the Sparkling Water assembly jar.

```
./bin/spark-shell --jars license.sig,mojo2-runtime.jar,sparkling-water-assembly.jar
```

At this point, you should have available a Sparkling Water interactive terminal where you can try out predictions. If
you would like to productionalize the scoring process, you can use the same configuration, except instead of using
./bin/spark-shell, you would use ./bin/spark-submit to submit your job to a cluster.

```scala
// First, specify the dependency
import org.apache.spark.ml.h2o.models.H2OMOJOPipelineModel
```

```scala
// Load the pipeline
val mojo = H2OMOJOPipelineModel.createFromMojo("file:///path/to/the/pipeline.mojo")

// This option ensures that the output columns are named properly. If you want to use
// old behaviour
// when all output columns were stored inside an array, don’t specify this
// configuration option
// or set it to False. We however strongly encourage users to set this to true as
// below.
mojo.setNamedMojoOutputColumns(true)
```

```scala
// Load the data as Spark's Data Frame
val dataFrame = spark.read.option("header", "true").csv("file:///path/to/the/data.csv")
```
Run the predictions. The predictions contain all the original columns plus the predictions added as new columns.

```scala
val predictions = mojo.transform(dataFrame)

// You can easily get the predictions for desired column using the helper function as follows:
predictions.select(mojo.selectPredictionUDF("AGE"))
```
DEPLOYING THE MOJO PIPELINE

Driverless AI can deploy the MOJO scoring pipeline for you to test and/or to integrate into a final product.

Notes:

• This section describes how to deploy a MOJO scoring pipeline and assumes that a MOJO scoring pipeline exists. Refer to the *Driverless AI MOJO Scoring Pipeline* section for information on how to build a MOJO scoring pipeline.

• This is an early feature that will eventually support multiple different deployments. At this point, Driverless AI can only deploy the trained MOJO scoring pipeline as an *AWS Lambda Function*, i.e., a server-less scorer running in Amazon Cloud and charged by the actual usage.

24.1 Deployments Overview Page

All of the MOJO scoring pipeline deployments are available in the Deployments Overview page, which is available from the top menu. This page lists all active deployments and the information needed to access the respective end-points. In addition, it allows you to stop any deployments that are no longer needed.

24.2 Amazon Lambda Deployment

24.2.1 Driverless AI Prerequisites

• Driverless AI MOJO Scoring Pipeline: To deploy a MOJO scoring pipeline as an AWS lambda function, the MOJO pipeline archive has to be created first by choosing the *Build MOJO Scoring Pipeline* option on the completed experiment page. Refer to the *Driverless AI MOJO Scoring Pipeline* section for information on how to build a MOJO scoring pipeline.

• Terraform: In addition, the Terraform tool ([https://www.terraform.io/](https://www.terraform.io/)) has to be installed on the system running Driverless AI. The tool is included in the Driverless AI Docker images but not in native install packages. To install Terraform, please follow steps on *Terraform installation page*. **Note:** Terraform is not available on
every platform. In particular, there is no Power build, so AWS Lambda Deployment is currently not supported on Power installations of Driverless AI.

24.2.2 AWS Prerequisites

Usage Plans

Usage plans must be enabled in the target AWS region in order for API keys to work when accessing the AWS Lambda via its REST API. Refer to https://aws.amazon.com/blogs/aws/new-usage-plans-for-amazon-api-gateway/ for more information.

Access Permissions

The following AWS access permissions need to be provided to the role in order for Driverless AI Lambda deployment to succeed.

- AWSLambdaFullAccess
- IAMFullAccess
- AmazonAPIGatewayAdministrator

The policy can be further stripped down to restrict Lambda and S3 rights using the JSON policy definition as follows:

```json
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Sid": "VisualEditor0",
         "Effect": "Allow",
         "Action": [
            "iam:GetPolicyVersion",
            "iam:DeletePolicy",
            "iam:CreateRole",
            "iam:AttachRolePolicy",
            "iam:ListInstanceProfilesForRole",
            "iam:PassRole",
            "iam:DetachRolePolicy",
            "iam:ListAttachedRolePolicies",
            "iam:GetRole",
            "iam:GetPolicy",
            "iam:DeleteRole",
            "iam:CreatePolicy",
            "iam:ListPolicyVersions"
       ]
      }
   ]
}
```
(continues on next page)
24.2.3 Deploying on Amazon Lambda

Once the MOJO pipeline archive is ready, Driverless AI provides a **Deploy** option on the completed experiment page.

**Notes:**

- This button is only available after the MOJO Scoring Pipeline has been built.
- This button is not available on PPC64LE environments.
This option opens a new dialog for setting the AWS account credentials (or use those supplied in the Driverless AI configuration file or environment variables), AWS region, and the desired deployment name (which must be unique per Driverless AI user and AWS account used).

Amazon Lambda deployment parameters:

- **Deployment Name**: A unique name of the deployment. By default, Driverless AI offers a name based on the
name of the experiment and the deployment type. This has to be unique both for Driverless AI user and the AWS account used.

- **Region**: The AWS region to deploy the MOJO scoring pipeline to. It makes sense to choose a region geographically close to any client code calling the endpoint in order to minimize request latency. (See also AWS Regions and Availability Zones.)

- **Use AWS environment variables**: If enabled, the AWS credentials are taken from the Driverless AI configuration file (see records `deployment_aws_access_key_id` and `deployment_aws_secret_access_key`) or environment variables (`DRIVERLESS_AI_DEPLOYMENT_AWS_ACCESS_KEY_ID` and `DRIVERLESS_AI_DEPLOYMENT_AWS_SECRET_ACCESS_KEY`). This would usually be entered by the Driverless AI installation administrator.

- **AWS Access Key ID** and **AWS Secret Access Key**: Credentials to access the AWS account. This pair of secrets identifies the AWS user and the account and can be obtained from the AWS account console.

### 24.2.4 Testing the Lambda Deployment

On a successful deployment, all the information needed to access the new endpoint (URL and an API Key) is printed, and the same information is available in the **Deployments Overview Page** after clicking on the deployment row.

Note that the actual scoring endpoint is located at the path `/score`. In addition, to prevent DDoS and other malicious activities, the resulting AWS lambda is protected by an API Key, i.e., a secret that has to be passed in as a part of the request using the `x-api-key` HTTP header.

The request is a JSON object containing attributes:

- **fields**: A list of input column names that should correspond to the training data columns.

- **rows**: A list of rows that are in turn lists of cell values to predict the target values for.
• optional includeFieldsInOutput: A list of input columns that should be included in the output.

An example request providing 2 columns on the input and asking to get one column copied to the output looks as follows:

```
{
   "fields": [ 
     "age", "salary"
   ],
   "includeFieldsInOutput": [ 
     "salary"
   ],
   "rows": [ 
     [ 
       "48.0", "15000.0"
     ],
     [ 
       "35.0", "35000.0"
     ],
     [ 
       "18.0", "22000.0"
     ]
   ]
}
```

Assuming the request is stored locally in a file named test.json, the request to the endpoint can be sent, e.g., using the curl utility, as follows:

```
$ URL={place the endpoint URL here}
$ API_KEY={place the endpoint API key here}
$ curl \
   -d @test.json \
   -X POST \ 
   -H "x-api-key: ${API_KEY}" \ 
   ${URL}/score
```

The response is a JSON object with a single attribute score, which contains the list of rows with the optional copied input values and the predictions.

For the example above with a two class target field, the result is likely to look something like the following snippet. The particular values would of course depend on the scoring pipeline:

```
{
   "score": [ 
     [ 
       "48.0",
       "0.6240277982943945",
       "0.045458571508101536",
     ],
     [ 
       "35.0",
       "0.7209441819603676",
       "0.06299909138586585",
     ],
     [ 
       "18.0",
       "0.7209441819603676",
       "0.06299909138586585",
     ]
   ]
}
```

(continues on next page)
24.3 AWS Deployment Issues

We create a new S3 bucket per AWS Lambda deployment. The bucket names have to be unique throughout AWS S3, and one user can create a maximum of 100 buckets. Therefore, we recommend setting the bucket name used for deployment with the `deployment_aws_bucket_name` config option.
H2O Driverless AI is an automatic machine learning platform that uses feature engineering recipes from some of the world’s best data scientists to deliver highly accurate machine learning models. As part of the automatic feature engineering process, the system uses a variety of transformers to enhance the available data. This section describes what’s happening underneath the hood, including details about the feature engineering transformations and time series and natural language processing functionality.

Refer to one of the following topics:

- Data Sampling
- Driverless AI Transformations
- Internal Validation Technique
- Time Series in Driverless AI
- NLP in Driverless AI
Driverless AI does not perform any type of down sampling unless the dataset is big. What is considered big is dependent on your accuracy setting and the `statistical_threshold_data_size_large` parameter in the `config.toml` or in the Expert Settings. You can see if the data will be sampled by viewing the Experiment Preview when you set up the experiment. In the experiment preview below, I can see that my data was sampled down to 5 million rows.
If Driverless AI decides to sample the data based on these settings and the data size, then Driverless AI will perform the following types of sampling:

- Random sampling for regression problems and binary problems that are not considered imbalanced
- Stratified sampling for multi-class problems
- Imbalanced sampling for binary problems where the data is considered imbalanced
  - By default, imbalanced is defined as when the majority class is 5 times more common than the minority class. (This is also configurable.)

With imbalanced sampling, there are two approaches:
- Undersampling of the majority class
- Quantile imbalanced sampling

Quantile imbalanced sampling is not turned on by default but you can enable it in the Expert Settings. Quantile imbalanced sampling takes all of the minority class records and takes only a sample of the majority class.

The steps for Quantile Imbalanced Sampling are shown below:

1. Train a preliminary model on a subset of data to predict the target column.
2. Assign each record in the data a probability.
3. Bin the probabilities into deciles for the records from the majority class.
4. Sample the records from the majority class from each decile bin. This will ensure that the distribution of the predicted probability from the sample majority class is smooth.

Generally, we do not want to perform data sampling unless the dataset is really large. We have found that imbalanced sampling does not necessarily improve the results. You can always use the weight column if you want the majority class to be weighted more heavily in the model and your dataset is not large.
Transformations in Driverless AI are applied to columns in the data. The transformers create the engineered features in experiments.

Driverless AI provides a number of transformers. The downloaded experiment logs include the transformations that were applied to your experiment. Note that you can exclude transformations in the `config.toml` file, and that list of excluded transformers will also be available in the experiment log.

### 27.1 Available Transformers

The following transformers are available for classification (multiclass and binary) and regression experiments.

#### 27.1.1 Numeric Transformers (Integer, Real, Binary)

- **BulkInteractionsTransformer**
  
  The Bulk Interactions Transformer add, divide, multiply, and subtract two numeric columns in the data to create a new feature.

- **ClusterDistTransformer**
  
  The Cluster Distance Transformer clusters selected numeric columns and uses the distance to a specific cluster as a new feature.

- **ClusterTETransformer**
  
  The Cluster Target Encoding Transformer clusters selected numeric columns and calculates the mean of the response column for each cluster. The mean of the response is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **NumCatTETransformer**
  
  The Numeric Categorical Target Encoding Transformer calculates the mean of the response column for several selected columns. If one of the selected columns is numeric, it is first converted to categorical by binning. The mean of the response column is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **NumToCatTETransformer**
  
  The Numeric to Categorical Target Encoding Transformer converts numeric columns to categoricals by binning and then calculates the mean of the response column for each group. The mean of the response for the bin is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **NumToCatWoEMonotonicTransformer**
The Numeric to Categorical Weight of Evidence Monotonic Transformer converts a numeric column to categorical by binning and then calculates Weight of Evidence for each bin. The monotonic constraint ensures the bins of values are monotonically related to the Weight of Evidence value. The Weight of Evidence is used as a new feature. Weight of Evidence measures the “strength” of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.

- **NumToCatWoETransformer**

  The Numeric to Categorical Weight of Evidence Transformer converts a numeric column to categorical by binning and then calculates Weight of Evidence for each bin. The Weight of Evidence is used as a new feature. Weight of Evidence measures the “strength” of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.

- **TruncSVDNumTransformer**

  Truncated SVD Transformer trains a Truncated SVD model on selected numeric columns and uses the components of the truncated SVD matrix as new features.

### 27.1.2 Time Series Experiments Transformers

- **LagsTransformer**

  The Lags Transformer creates target/feature lags, possibly over groups. Each lag is used as a new feature. Lag transformers may apply to categorical (strings) features or binary/multiclass string valued targets after they have been internally numerically encoded.

- **LagsAggregatesTransformer**

  The Lags Aggregates Transformer calculates aggregations of target/feature lags like mean(lag7, lag14, lag21) with support for mean, min, max, median, sum, skew, kurtosis, std. The aggregation is used as a new feature.

- **EwmaLagsTransformer**

  The Exponentially Weighted Moving Average (EWMA) Transformer calculates the exponentially weighted moving average of target or feature lags.

- **LagsInteractionTransformer**

  The Lags Interaction Transformer creates target/feature lags and calculates interactions between the lags (lag2 - lag1, for instance). The interaction is used as a new feature.

### 27.1.3 Categorical Transformers (String)

- **CVCatNumEncode**

  The Cross Validation Categorical to Numeric Encoding Transformer calculates an aggregation of a numeric column for each value in a categorical column (ex: calculate the mean Temperature for each City) and uses this aggregation as a new feature.

- **CVTargetEncode**

  The Cross Validation Target Encoding Transformer calculates the mean of the response column for each value in a categorical column and uses this as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **FrequentTransformer**
The Frequent Transformer calculates the frequency for each value in categorical column(s) and uses this as a new feature. This count can be either the raw count or the normalized count.

- **LexiLabelEncoder**
  
  The Lexi Label Encoder sorts a categorical column in lexicographical order and uses the order index created as a new feature.

- **NumCatTETransformer**
  
  The Numeric Categorical Target Encoding Transformer calculates the mean of the response column for several selected columns. If one of the selected columns is numeric, it is first converted to categorical by binning. The mean of the response column is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **OHETransformer**
  
  The One-hot Encoding transformer converts a categorical column to a series of boolean features by performing one-hot encoding. The boolean features are used as new features.

- **SortedLETransformer**
  
  The Sorted Label Encoding Transformer sorts a categorical column by the response column and uses the order index created as a new feature.

- **WeightOfEvidenceTransformer**
  
  The Weight of Evidence Transformer calculates Weight of Evidence for each value in categorical column(s). The Weight of Evidence is used as a new feature. Weight of Evidence measures the “strength” of a grouping for separating good and bad risk and is calculated by taking the log of the ratio of distributions for a binary response column.

  \[
  WOE = \ln \left( \frac{\text{Distribution of Goods}}{\text{Distribution of Bads}} \right)
  \]

  This only works with a binary target variable. The likelihood needs to be created within a stratified kfold if a fit_transform method is used. More information can be found here: [http://ucanalytics.com/blogs/information-value-and-weight-of-evidencebanking-case/](http://ucanalytics.com/blogs/information-value-and-weight-of-evidencebanking-case/).

### 27.1.4 Text Transformers (String)

- **TextBiGRUTransformer**
  
  Trains a bi-directional GRU Tensorflow model on word embeddings created from a text feature to predict the response column. The GRU prediction is used as a new a feature. Cross Validation is used when training the GRU model to prevent overfitting.

- **TextCharCNNTransformer**
  
  Trains a CNN Tensorflow model on character embeddings created from a text feature to predict the response column. The CNN prediction is used as a new a feature. Cross Validation is used when training the CNN model to prevent overfitting.

- **TextClustDistTransformer**
The Text Cluster Distance Transformer clusters a TF-IDF matrix created from a text feature and uses the distance to a specific cluster as a new feature.

- **TextClustTETransformer**
  The Text Cluster Target Encoding Transformer clusters a TF-IDF matrix created from a text feature. The mean of the response is calculated for each cluster and this is used as a new feature. Cross Validation is used to calculate mean response to prevent overfitting.

- **TextCNNTransformer**
  The Text CNN Transformer trains a CNN Tensorflow model on word embeddings created from a text feature to predict the response column. The CNN prediction is used as a new feature. Cross Validation is used when training the CNN model to prevent overfitting.

- **TextLinModelTransformer**
  The Text Linear Model Transformer trains a linear model on a TF-IDF matrix created from a text feature to predict the response column. The linear model prediction is used as a new feature. Cross Validation is used when training the linear model to prevent overfitting.

- **TextTransformer**
  The Text Transformer tokenizes a text column and creates a TFIDF matrix (term frequency-inverse document frequency) or count (count of the word) matrix. This may be followed by dimensionality reduction using truncated SVD. Selected components of the TF-IDF/Count matrix are used as new features.

### 27.1.5 Time Transformers (Date, Time)

- **DatesTransformer**
  The Dates Transformer retrieves any date values, including:
  - Year
  - Quarter
  - Month
  - Day
  - Day of year
  - Week
  - Week day
  - Hour
  - Minute
  - Second

- **IsHolidayTransformer**
  The Is Holiday Transformer determines if a date column is a holiday. A boolean column indicating if the date is a holiday is added as a new feature. Creates a separate feature for holidays in the United States, United Kingdom, Germany, Mexico, and the European Central Bank. Other countries available in the python Holiday package can be added via the configuration file.
27.2 Example Transformations

In this section, we will describe some of the available transformations using the example of predicting house prices on the example dataset.

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>$700K</td>
</tr>
</tbody>
</table>

27.2.1 Frequent Transformer

- the count of each categorical value in the dataset
- the count can be either the raw count or the normalized count

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>Freq_State</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>4,500</td>
</tr>
</tbody>
</table>

There are 4,500 properties in this dataset with state = NY.

27.2.2 Bulk Interactions Transformer

- add, divide, multiply, and subtract two columns in the data

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>Interaction_NumBeds#subtract#NumBaths</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>1</td>
</tr>
</tbody>
</table>

There is one more bedroom than there are number of bathrooms for this property.

27.2.3 Truncated SVD Numeric Transformer

- truncated SVD trained on selected numeric columns of the data
- the components of the truncated SVD will be new features

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>TruncSVD_Price_NumBeds_NumBaths_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>0.632</td>
</tr>
</tbody>
</table>

The first component of the truncated SVD of the columns Price, Number of Beds, Number of Baths.

27.2.4 Dates Transformer

- get year, get quarter, get month, get day, get day of year, get week, get week day, get hour, get minute, get second
Using Driverless AI, Release 1.6.2

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>DateBuilt_Month</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>1</td>
</tr>
</tbody>
</table>

The home was built in the month January.

### 27.2.5 Text Transformer

- transform text column using methods: TFIDF or count (count of the word)
- this may be followed by dimensionality reduction using truncated SVD

### 27.2.6 Categorical Target Encoding Transformer

- cross validation target encoding done on a categorical column

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>CV_TE_State</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>550,000</td>
</tr>
</tbody>
</table>

The average price of properties in NY state is $550,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.

### 27.2.7 Numeric to Categorical Target Encoding Transformer

- numeric column converted to categorical by binning
- cross validation target encoding done on the binned numeric column

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>CV_TE_SquareFootage</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>345,000</td>
</tr>
</tbody>
</table>

The column *Square Footage* has been bucketed into 10 equally populated bins. This property lies in the *Square Footage* bucket 1,572 to 1,749. The average price of properties with this range of square footage is $345,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.

### 27.2.8 Cluster Target Encoding Transformer

- selected columns in the data are clustered
- target encoding is done on the cluster ID

<table>
<thead>
<tr>
<th>Date Built</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>ClusterTE_4_NumBeds_NumBaths_SquareFootage</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>450,000</td>
</tr>
</tbody>
</table>

The columns: *Num Beds, Num Baths, Square Footage* have been segmented into 4 clusters. The average price of properties in the same cluster as the selected property is $450,000*.

*In order to prevent overfitting, Driverless AI calculates this average on out-of-fold data using cross validation.
27.2.9 Cluster Distance Transformer

- selected columns in the data are clustered
- the distance to a chosen cluster center is calculated

<table>
<thead>
<tr>
<th>Date</th>
<th>Square Footage</th>
<th>Num Beds</th>
<th>Num Baths</th>
<th>State</th>
<th>Price</th>
<th>ClusterDist_4_NumBeds_NumBaths_SquareFootage_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/1920</td>
<td>1700</td>
<td>3</td>
<td>2</td>
<td>NY</td>
<td>700,000</td>
<td>0.83</td>
</tr>
</tbody>
</table>

The columns: Num Beds, Num Baths, Square Footage have been segmented into 4 clusters. The difference from this record to Cluster 1 is 0.83.
This section describes the technique behind internal validation in Driverless AI.

For the experiment, Driverless AI will either:

1. split the data into a training set and internal validation set
   or
2. use cross validation to split the data into \( n \) folds

Driverless AI chooses the method based on the size of the data and the Accuracy setting. For method 1, part of the data is removed to be used for internal validation. (*Note:* This train and internal validation split may be repeated if the data is small so that more data can be used for training.)

For method 2, however, no data is wasted for internal validation. With cross validation, the whole dataset is utilized, and each model is trained on a different subset of the training data. The following visualization shows an example of cross validation with 5 folds.

![Cross Validation Visualization]

Driverless AI randomly splits the data into the specified number of folds for cross validation. With cross validation, the whole dataset is utilized, and each model is trained on a different subset of the training data.

Driverless AI will not automatically create the internal validation data randomly if a user provides a Fold Column or a Validation Dataset. If a Fold Column or a Validation Dataset is provided, Driverless AI will use that data to calculate the performance of the Driverless AI models and to calculate all performance graphs and statistics.

If the experiment is a Time Series use case, and a Time Column is selected, Driverless AI will change the way the internal validation data is created. In the case of temporal data, it is important to train on historical data and validate on more recent data. Driverless AI does not perform random splits, but instead respects the temporal nature of the data.
to prevent any data leakage. In addition, the train/validation split is a function of the time gap between train and test as well as the forecast horizon (amount of time periods to predict). If test data is provided, Driverless AI will suggest values for these parameters that lead to a validation set that resembles the test set as much as possible. But users can control the creation of the validation split in order to adjust it to the actual application.
TIME SERIES IN DRIVERLESS AI

Time-series forecasting is one of the most common and important tasks in business analytics. There are many real-world applications like sales, weather, stock market, energy demand, just to name a few. At H2O, we believe that automation can help our users deliver business value in a timely manner. Therefore, we combined advanced time series analysis and our Kaggle Grand Masters’ time-series recipes into Driverless AI.

The key features/recipes that make automation possible are:

- Automatic handling of time groups (e.g., different stores and departments)
- Robust time-series validation
  - Accounts for gaps and forecast horizon
  - Uses past information only (i.e., no data leakage)
- Time-series-specific feature engineering recipes
  - Date features like day of week, day of month, etc.
  - AutoRegressive features, like optimal lag and lag-features interaction
  - Different types of exponentially weighted moving averages
  - Aggregation of past information (different time groups and time intervals)
  - Target transformations and differentiation
- Integration with existing feature engineering functions (recipes and optimization)
- Automatic pipeline generation (See “From Kaggle Grand Masters’ Recipes to Production Ready in a Few Clicks” blog post.)

29.1 Understanding Time Series

29.1.1 Modeling Approach

Driverless AI uses GBMs, GLMs and neural networks with a focus on time-series-specific feature engineering. The feature engineering includes:

- Autoregressive elements: creating lag variables
- Aggregated features on lagged variables: moving averages, exponential smoothing descriptive statistics, correlations
- Date-specific features: week number, day of week, month, year
- Target transformations: Integration/Differentiation, univariate transforms (like logs, square roots)
This approach is combined with AutoDL features as part of the genetic algorithm. The selection is still based on validation accuracy. In other words, the same transformations/genes apply; plus there are new transformations that come from time series. Some transformations (like target encoding) are deactivated.

When running a time-series experiment, Driverless AI builds multiple models by rolling the validation window back in time (and potentially using less and less training data).

### 29.1.2 User-Configurable Options

**Gap and Horizon**

The guiding principle for properly modeling a time series forecasting problem is to use the historical data in the model training dataset such that it mimics the data/information environment at scoring time (i.e. deployed predictions). Specifically, you want to partition the training set to account for: 1) the information available to the model when making predictions and 2) the length of predictions to make.

Given a training dataset, gap and prediction length are parameters that determine how to split the training dataset into training samples and validation samples.

**Gap** is the amount of missing time bins between the end of a training set and the start of test set (with regards to time). For example:

- Assume you have daily data with days 1/1, 1/2, 1/3, 1/4 in train.
- The corresponding time bins would be 1, 2, 3, 4 for a time period of 1 day.
- Given that, the first valid time bin to predict is 5.
- As a result, Gap = max(time bin train) - min(time bin test) - 1.

Quite often, it is not possible to have the most recent data available when applying a model (or it is costly to update the data table too often); hence models need to be built accounting for a “future gap”. For example if it takes a week to update a certain data table, ideally we would like to predict “7 days ahead” with the data as it is “today”; hence a gap of 7 days would be sensible. Not specifying a gap and predicting 7 days ahead with the data as it is 7 days ahead is unrealistic (and can cannot happen as we update the data on a weekly basis in this example).

Similarly, gap can be used for those who want to forecast further in advance. For example, users want to know what will happen 7 days in the future, they will set the gap to 7 days.

**Horizon** (or prediction length) is the period that the test data spans for (for example, one day, one week, etc.). In other words it is the future period that the model can make predictions for.
The periodicity of updating the data may require model predictions to account for significant time in the future. In an ideal world where data can be updated very quickly, predictions can always be made having the most recent data available. In this scenario there is no need for a model to be able to predict cases that are well into the future, but rather focus on maximizing its ability to predict short term. However this is not always the case, and a model needs to be able to make predictions that span deep into the future because it may be too costly to make predictions every single day after the data gets updated.

In addition, each future data point is not the same. For example, predicting tomorrow with today’s data is easier than predicting 2 days ahead with today’s data. Hence specifying the horizon can facilitate building models that optimize prediction accuracy for these future time intervals.

**Groups**

Groups are categorical columns in the data that can significantly help predict the target variable in time series problems. For example, one may need to predict sales given information about stores and products. Being able to identify that the combination of store and products can lead to very different sales is key for predicting the target variable, as a big store or a popular product will have higher sales than a small store and/or with unpopular products.

For example, if we don’t know that the store is available in the data, and we try to see the distribution of sales along time (with all stores mixed together), it may look like that:

The same graph grouped by store gives a much clearer view of what the sales look like for different stores.
Lag

The primary generated time series features are lag features, which are a variable’s past values. At a given sample with time stamp $t$, features at some time difference $T$ (lag) in the past are considered. For example, if the sales today are 300, and sales of yesterday are 250, then the lag of one day for sales is 250. Lags can be created on any feature as well as on the target.

As previously noted, the training dataset is appropriately split such that the amount of validation data samples equals that of the testing dataset samples. If we want to determine valid lags, we must consider what happens when we will evaluate our model on the testing dataset. Essentially, the minimum lag size must be greater than the gap size.

Aside from the minimum useable lag, Driverless AI attempts to discover predictive lag sizes based on autocorrelation.
“Lagging” variables are important in time series because knowing what happened in different time periods in the past can greatly facilitate predictions for the future. Consider the following example to see the lag of 1 and 2 days:

<table>
<thead>
<tr>
<th>Date</th>
<th>Sales</th>
<th>Lag1</th>
<th>Lag2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2018</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2/1/2018</td>
<td>150</td>
<td>100</td>
<td>-</td>
</tr>
<tr>
<td>3/1/2018</td>
<td>160</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>4/1/2018</td>
<td>200</td>
<td>160</td>
<td>150</td>
</tr>
<tr>
<td>5/1/2018</td>
<td>210</td>
<td>200</td>
<td>160</td>
</tr>
<tr>
<td>6/1/2018</td>
<td>150</td>
<td>210</td>
<td>200</td>
</tr>
<tr>
<td>7/1/2018</td>
<td>160</td>
<td>150</td>
<td>210</td>
</tr>
<tr>
<td>8/1/2018</td>
<td>120</td>
<td>160</td>
<td>150</td>
</tr>
<tr>
<td>9/1/2018</td>
<td>80</td>
<td>120</td>
<td>160</td>
</tr>
<tr>
<td>10/1/2018</td>
<td>70</td>
<td>80</td>
<td>120</td>
</tr>
</tbody>
</table>

29.1.3 Settings Determined by Driverless AI

Window/Moving Average

Using the above Lag table, a moving average of 2 would constitute the average of Lag1 and Lag2:

<table>
<thead>
<tr>
<th>Date</th>
<th>Sales</th>
<th>Lag1</th>
<th>Lag2</th>
<th>MA2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/1/2018</td>
<td>100</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2/1/2018</td>
<td>150</td>
<td>100</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>3/1/2018</td>
<td>160</td>
<td>150</td>
<td>100</td>
<td>125</td>
</tr>
<tr>
<td>4/1/2018</td>
<td>200</td>
<td>160</td>
<td>150</td>
<td>155</td>
</tr>
<tr>
<td>5/1/2018</td>
<td>210</td>
<td>200</td>
<td>160</td>
<td>180</td>
</tr>
<tr>
<td>6/1/2018</td>
<td>150</td>
<td>210</td>
<td>200</td>
<td>205</td>
</tr>
<tr>
<td>7/1/2018</td>
<td>160</td>
<td>150</td>
<td>210</td>
<td>180</td>
</tr>
<tr>
<td>8/1/2018</td>
<td>120</td>
<td>160</td>
<td>150</td>
<td>155</td>
</tr>
<tr>
<td>9/1/2018</td>
<td>80</td>
<td>120</td>
<td>160</td>
<td>140</td>
</tr>
<tr>
<td>10/1/2018</td>
<td>70</td>
<td>80</td>
<td>120</td>
<td>100</td>
</tr>
</tbody>
</table>

Aggregating multiple lags together (instead of just one) can facilitate stability for defining the target variable. It may include various lags values, for example lags [1-30] or lags [20-40] or lags [7-70 by 7].

Exponential Weighting

Exponential weighting is a form of weighted moving average where more recent values have higher weight than less recent values. That weight is exponentially decreased over time based on an alpha (a) (hyper) parameter (0,1), which is normally within the range of [0.9 - 0.99]. For example:

- Exponential Weight = a**(time)
- If sales 1 day ago = 3.0 and 2 days ago =4.5 and a=0.95:
- Exp. smooth = 3.0*(0.95**1) + 4.5*(0.95**2) / ((0.95**1) + (0.95**2)) =3.73 approx.
29.2 Time Series Constraints

29.2.1 Dataset Size

Usually, the forecast horizon (prediction length) $H$ equals the number of time periods in the testing data $N_{TEST}$ (i.e. $N_{TEST} = H$). You want to have enough training data time periods $N_{TRAIN}$ to score well on the testing dataset. At a minimum, the training dataset should contain at least three times as many time periods as the testing dataset (i.e. $N_{TRAIN} \geq 3N_{TEST}$). This allows for the training dataset to be split into a validation set with the same amount of time periods as the testing dataset while maintaining enough historical data for feature engineering.

29.3 Time Series Use Case: Sales Forecasting

Below is a typical example of sales forecasting based on the Walmart competition on Kaggle. In order to frame it as a machine learning problem, we formulate the historical sales data and additional attributes as shown below:

**Raw data**

<table>
<thead>
<tr>
<th>Store</th>
<th>Department</th>
<th>Date</th>
<th>Weekly Sales</th>
<th>Mark Down 1</th>
<th>Mark Down 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2/5/10</td>
<td>$24,924.50</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2/5/10</td>
<td>$50,605.27</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>2/5/10</td>
<td>$13,740.12</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>2/5/10</td>
<td>$39,954.04</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

**Data formulated for machine learning**

<table>
<thead>
<tr>
<th>Store</th>
<th>Department</th>
<th>Mark Down 1</th>
<th>Mark Down 2</th>
<th>Weekly Sales 2 Weeks Ago</th>
<th>Weekly Sales Last Week</th>
<th>Weekly Sales Next Week</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>NA</td>
<td>$24,924.50</td>
<td>$46,039.49</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>$24,924.50</td>
<td>$46,039.49</td>
<td>$41,595.55</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>$46,039.49</td>
<td>$41,595.55</td>
<td>$19,403.54</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>$41,595.55</td>
<td>$19,403.54</td>
<td>$21,827.90</td>
</tr>
</tbody>
</table>

The additional attributes are attributes that we will know at time of scoring. In this example, we want to forecast the next week of sales. Therefore, all of the attributes included in our data must be known at least one week in advance. In this case, we assume that we will know whether or not a Store and Department will be running a promotional markdown. We will not use features like the temperature of the Week since we will not have that information at the time of scoring.
Once you have your data prepared in tabular format (see raw data above), Driverless AI can formulate it for machine learning and sort out the rest. If this is your very first session, the Driverless AI assistant will guide you through the journey.

Similar to previous Driverless AI examples, you need to select the dataset for training/test and define the target. For time-series, you need to define the time column (by choosing AUTO or selecting the date column manually). If weighted scoring is required (like the Walmart Kaggle competition), you can select the column with specific weights for different samples.

If you prefer to use automatic handling of time groups, you can leave the setting for time groups columns as AUTO.
Expert users can define specific time groups and change other settings as shown below. The Driverless AI time series expert settings provide a matrix of gap and prediction length combinations to choose from. The options for prediction length is based on quantiles of valid training dataset splits. Also, notice that the maximum prediction length (39 weeks) is set to exactly the size of the testing dataset. Because Driverless AI attempts to auto-detect the gap based on the training and testing datasets, it indentifies a better split (i.e. gap and prediction length combination) by the brightness of the matrix cells.

Once the experiment is finished, you can make new predictions and download the scoring pipeline just like any other Driverless AI experiments.

### 29.4 Time Series EXPERT SETTINGS

The user may further configure the time series experiments with a dedicated set of options available through the EXPERT SETTINGS. The EXPERT SETTINGS’ panel is available from within the experiment page right above the INTERPRETABILITY radar. This is a different panel than the one referenced above and can be accessed via:
List of Time Series Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>lag-based recipe</td>
<td>If <strong>enabled</strong> Driverless AI will attempt to generate lags other autoregressive features as part of the feature engineering process. If <strong>disabled</strong>, feature engineering will be limited to tabular data. In both cases the validation schema will always have models built on past data and evaluated on future data.</td>
</tr>
<tr>
<td>lags override</td>
<td>A list of lags (as integers) may be provided to be used when generating time series features. For example if <code>[1,5,10]</code> is passed, only lags of 1,5,10 will be considered.</td>
</tr>
<tr>
<td>probability to create non-target</td>
<td>When lag features are evaluated from Driverless AI's genetic algorithm, they are most commonly derived from the lag features target variable. However on a small probability, Driverless AI will also consider lagging other variables too. Increasing this probability would result in Driverless AI evaluating other variables (besides the target) more often in the feature engineering process.</td>
</tr>
<tr>
<td>Holiday Features</td>
<td>If <strong>enabled</strong> Driverless AI will generate some time series features based on whether a specific day is a bank holiday or not. This is currently based on US Bank holidays.</td>
</tr>
</tbody>
</table>

29.5 Using a Driverless AI Time Series Model to Forecast

When you set the experiment’s forecast horizon, you are telling the Driverless AI experiment the dates this model will be asked to forecast for. In the Walmart Sales example, we set the Driverless AI forecast horizon to 1 (1 week in the future). This means that Driverless AI expects this model to be used to forecast 1 week after training ends. Since the training data ends on 2012-10-26, then this model should be used to score for the week of 2012-11-02.

What should the user do once the 2012-11-02 week has passed?

There are two options:

1. **Trigger a Driverless AI experiment to be trained once the forecast horizon ends**
Using Driverless AI, Release 1.6.2

- a Driverless AI experiment will need to be re-trained every week

2. Use **Test Time Augmentation** to update historical features so that we can use the same model to forecast outside of the forecast horizon

**Test Time Augmentation** refers to the process where the model stays the same but the features are refreshed using the latest data. In our Walmart Sales Forecasting example, a feature that may be very important is the Weekly Sales from the previous week. Once we move outside of the forecast horizon, our model no longer knows the Weekly Sales from the previous week. By performing Test Time Augmentation, Driverless AI will automatically generate these historical features if new data is provided.

In Option 1, we would launch a new Driverless AI experiment every week with the latest data and use the resulting model to forecast the next week. In Option 2, we would continue using the same Driverless AI experiment outside of the forecast horizon by using Test Time Augmentation.

Both options have their advantages and disadvantages. By re-training an experiment with the latest data, Driverless AI has the ability to possibly improve the model by changing the features used, choosing a different algorithm, and/or selecting different parameters. As the data changes over time, for example, Driverless AI may find that the best algorithm for this use case has changed.

Using Test Time Augmentation to be able to continue using the same experiment over a longer period of time means there would be no need to continually repeat a model review process. The model may become out of date, however, and the MOJO scoring pipeline is not supported.

<table>
<thead>
<tr>
<th>Scoring Supported</th>
<th>Retraining Model</th>
<th>Test Time Augmentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driverless AI Scoring</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>Python Scoring Pipeline</td>
<td>Supported</td>
<td>Supported</td>
</tr>
<tr>
<td>MOJO Scoring Pipeline</td>
<td>Supported</td>
<td>Not Supported</td>
</tr>
</tbody>
</table>

For different use cases, there may be clear advantages for retraining an experiment after each forecast horizon or for using Test Time Augmentation. In this notebook, we show how to perform both and compare the performance: **Time Series Model Rolling Window**.

**How to trigger Test Time Augmentation?**

To tell Driverless AI to perform Test Time Augmentation, simply create your forecast data to include any data that occurred after the training data ended up to the date you want a forecast for. The date which you want Driverless AI to forecast should have NA where the target column is. Here is an example of forecasting 2012-11-09.

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Dept</th>
<th>Mark Down 1</th>
<th>Mark Down 2</th>
<th>Weekly_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-11-09</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>NA</td>
</tr>
</tbody>
</table>

If we do not include an NA in the Target column for the date we are interested in forecasting, then Test Time Augmentation will not be triggered.
Driverless AI version 1.3 introduced support for Natural Language Processing (NLP) experiments for text classification and regression problems. The Driverless AI platform has the ability to support both standalone text and text with other numerical values as predictive features. In particular, Driverless AI implements the following recipes and models.

**Text-specific feature engineering recipes:**
- TFIDF, Frequency of n-grams
- Truncated SVD
- Word embeddings

**Text-specific models to extract features from text:**
- Convolutional neural network models on word embeddings
- Linear models on TFIDF vectors

### 30.1 A Typical NLP Example: Sentiment Analysis

The following section provides an NLP example. This information is based on the Automatic Feature Engineering for Text Analytics blog post. A similar example using the Python Client is available in Appendix A: The Python Client.

This example uses a classical example of sentiment analysis on tweets using the US Airline Sentiment dataset from Figure Eight’s Data for Everyone library. We can split the dataset into training and test with this simple script. We will just use the tweets in the ‘text’ column and the sentiment (positive, negative or neutral) in the ‘airline_sentiment’ column for this demo. Here are some samples from the dataset:

<table>
<thead>
<tr>
<th>text</th>
<th>airline_sentiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>@JetBlue @roxydigital HAHA, you didn’t disappoint. Well done.</td>
<td>positive</td>
</tr>
<tr>
<td>@USAirways will do. Hoping for a voucher for a future flight</td>
<td>positive</td>
</tr>
<tr>
<td>@SouthwestAir hello I was talking with @SouthwestVerity but it says</td>
<td>neutral</td>
</tr>
<tr>
<td>you moved here, they sent me a voucher but does not work can u help?</td>
<td></td>
</tr>
<tr>
<td>@AmericanAir when will tomorrows flight Cancelled Flightsations at</td>
<td>negative</td>
</tr>
<tr>
<td>Dfw for AA flights be posted? We are on 2424 at 7am from LAX!</td>
<td></td>
</tr>
<tr>
<td>@United 3 hrs searching for flls, find 1 on site &amp; can’t book</td>
<td>negative</td>
</tr>
<tr>
<td>the $ offered bc 1 seg isn’t really available. #lies #falseadvertising</td>
<td></td>
</tr>
<tr>
<td>@AmericanAir even with calls you haven’t been able to help us any</td>
<td>negative</td>
</tr>
</tbody>
</table>

Once we have our dataset ready in the tabular format, we are all set to use the Driverless AI. Similar to other problems in the Driverless AI setup, we need to choose the dataset, and then specify the target column (‘airline_sentiment’).
Because we don’t want to use any other columns in the dataset, we need to click on Dropped Cols, and then exclude everything but text as shown below:
Next, we will need to make sure TensorFlow is enabled for the experiment. We can go to **Expert Settings** and enable **TensorFlow Models**.

At this point, we are ready to launch an experiment. Text features will be automatically generated and evaluated during the feature engineering process. Note that some features such as TextCNN rely on TensorFlow models. We recommend using GPU(s) to leverage the power of TensorFlow and accelerate the feature engineering process.
Once the experiment is done, users can make new predictions and download the scoring pipeline just like any other Driverless AI experiments.
CHAPTER THIRTYONE

DRIVERLESS AI LOGS

This section describes how to access Driverless AI logs and includes information on which logs to send in the event of a failure.

31.1 Accessing Driverless AI Logs

Driverless AI provides a number of logs that can be retrieved while visualizing datasets, while an experiment is running, and after an experiment is completed.

31.1.1 While Visualizing Datasets

When running Autovisualization, you can access the Autoviz logs by clicking the Display Logs button on the Visualize Datasets page.
This page presents logs created while the dataset visualization was being performed. You can download the **vis-data-server.log** file by clicking the **Download Logs** button on this page. This file can be used to troubleshoot any issues encountered during dataset visualization.

### 31.1.2 While an Experiment is Running

While the experiment is running, you can access the logs by clicking on the **Log** button on the experiment screen. The **Log** button can be found in the CPU/Memory section.

Clicking on the **Log** button will present the experiment logs in real time. You can download these logs by clicking on the **Download Logs** button in the upper right corner.
Only the `h2oai_experiment.log` can be downloaded while the experiment is running (for example: `h2oai_experiment_tobosoru.log`). It will have the same information as the logs being presented in real time on the screen.

For troubleshooting purposes, it is best to view the complete `h2oai_experiment.log` (or `h2oai_experiment_anonymized.log`). This will be available after the experiment finishes, as described in the next section.

### 31.1.3 After an Experiment has Finished

If the experiment has finished, you can download the logs by clicking on the **Download Logs** button at the center of the experiment screen.
This will download a zip file which includes the following logs:

- **h2oai_experiment.log**: This is the log corresponding to the experiment.
- **h2oai_experiment_anonymized.log**: This is the log corresponding to the experiment where all data in the log is anonymized.
- **h2oai_server.log**: Contains the logs for all experiments and all users.
- **h2oai_server_anonymized.log**: Contains the logs for all experiments and all users where all data in the log is anonymized.
- **h2o.log**: This is the log corresponding to H2O-3. (H2O-3 is used internally for parts of Driverless AI.)

For troubleshooting, it is best to view the **h2oai_experiment.log** or **h2oai_experiment_anonymized.log**.

The following additional information about your particular experiment will also be included in the zip file:

- **tuning_leaderboard.txt**: The results of the parameter tuning stage. This contains the model parameters investigated and their performance.
- **gene_summary.txt**: A summary of the feature transformations available for each gene over the feature engineering iterations
- **features.txt**: The features used in the final Driverless AI model along with feature importance and feature description
- **details folder**: Contains standard streams for each of the subprocesses performed by Driverless AI. This information is for debugging purposes.

### 31.1.4 During Model Interpretation

Driverless AI allows you to view and download Python and/or Java logs while MLI is running. Note that these logs are not available for time-series experiments.
• The **Display MLI Python Logs** button allows you to view or download the Python log for the model interpretation. The downloaded file is named `h2oai_experiment_{mli_key}.log`.

• The **Display MLI Java Logs** button allows you to view or download the Java log for the model interpretation. The downloaded file is named `mli_experiment_{mli_key}.log`.

### 31.1.5 After Model Interpretation

You can view an MLI log for completed model interpretations by selecting the **Download MLI Logs** link on the MLI page.

This will download a zip file which includes the following logs:
• h2oai_experiment_{mli_key}.log: This is the log corresponding to the model interpretation.

• h2oai_experiment_{mli_key}_anonymized.log: This is the log corresponding to the model interpretation where all data in the log is anonymized.

• mli_experiment_{mli_key}.log: This is the Java log corresponding to the model interpretation. This file can be used to view logging information for successful interpretations. If MLI fails, then those logs are in ./tmp/h2oai_experiment_{mli_key}.log, ./tmp/h2oai_experiment_{mli_key}_anonymized.log, and ./tmp/mli_experiment_{mli_key}.log.

31.2 Sending Logs to H2O

This section describes the logs to send in the event of failures when running Driverless AI.

31.2.1 Dataset Failures

• Adding Datasets: If a dataset fails to import, a message on the screen should provide the reason for the failure. The logs to send are available in the Driverless AI ./tmp folder.

• Dataset Details: If a failure occurs when attempting to view Dataset Details, the logs to send are available in the Driverless AI ./tmp folder.

• Autovisualization: If a failure occurs when attempting to Visualize Datasets, a message on the screen should provide a reason for the failure. The logs to send are available in the Driverless AI ./tmp folder.

31.2.2 Experiments

• While Running an Experiment: As indicated previously, a Log button is available on the Experiment page. Clicking on the Log button will present the experiment logs in real time. You can download these logs by clicking on the Download Logs button in the upper right corner. You can also retrieve the h2oai_experiment.log for the corresponding experiment in the Driverless AI ./tmp folder.

31.2.3 MLI

• During Model Interpretation: If a failure occurs during model interpretation, then the logs to send are ./tmp/h2oai_experiment_{mli_key}.log and ./tmp/h2oai_experiment_{mli_key}_anonymized.log.
H2O Driverless AI is an artificial intelligence (AI) platform for automatic machine learning. Driverless AI automates some of the most difficult data science and machine learning workflows such as feature engineering, model validation, model tuning, model selection and model deployment. It aims to achieve highest predictive accuracy, comparable to expert data scientists, but in much shorter time thanks to end-to-end automation. Driverless AI also offers automatic visualizations and machine learning interpretability (MLI). Especially in regulated industries, model transparency and explanation are just as important as predictive performance. Modeling pipelines (feature engineering and models) are exported (in full fidelity, without approximations) both as Python modules and as pure Java standalone scoring artifacts.

This section provides answers to frequently asked questions. If you have additional questions about using Driverless AI, post them on Stack Overflow using the driverless-ai tag at http://stackoverflow.com/questions/tagged/driverless-ai.

**General**
- How is Driverless AI different than any other black box ML algorithm?
- How often do new versions come out?

**Installation/Upgrade/Authentication**
- How can I change my username and password?
- Can Driverless AI run on CPU-only machines?
- How can I upgrade to a newer version of Driverless AI?
- What kind of authentication is supported in Driverless AI?
- How can I automatically turn on persistence each time the GPU system reboots?
- How can I start Driverless AI on a different port that 12345?
- Can I set up SSL on Driverless AI?
- I received a “Must have exactly one OpenCL platform ’NVIDIA CUDA’” error. How can I fix that?
- Is it possible for multiple users to share a single Driverless AI instance?
- Can multiple Driverless AI users share a GPU server?
- How can I retrieve a list of Driverless AI users?
- Start of Driverless AI fails on the message “Segmentation fault (core dumped)” on Ubuntu 18/RHEL 7.6. How can I fix this?

**Data/Experiments/Predictions**
- Is there a file size limit for datasets?
- How much memory does Driverless AI require in order to run experiments?
Using Driverless AI, Release 1.6.2

- How many columns can Driverless AI handle?
- How should I use Driverless AI if I have large data?
- How does Driverless AI detect the ID column?
- Can Driverless AI handle data with missing values/nulls?
- How are outliers handled?
- If I drop several columns from the Train dataset, will Driverless AI understand that it needs to drop the same columns from the Test dataset?
- Does Driverless AI treat numeric variables as categorical variables?
- Which algorithms are used in Driverless AI?
- How can we turn on TensorFlow Neural Networks so they are evaluated?
- Does Driverless AI standardize the data?
- What objective function is used in XGBoost?
- Does Driverless AI perform internal or external validation?
- How does Driverless AI prevent overfitting?
- How does Driverless AI avoid the multiple hypothesis (MH) problem?
- How does Driverless AI suggest the experiment settings?
- What happens when I set Interpretability and Accuracy to the same number?
- Can I specify the number of GPUs to use when running Driverless AI?
- How can I create the simplest model in Driverless AI?
- Why is my experiment suddenly slow?
- When I run multiple experiments with different seeds, why do I see different scores, runtimes, and sizes on disk in the Experiments listing page?
- Why does the final model performance appear to be worse than previous iterations?
- How can I find features that may be causing data leakages in my Driverless AI model?
- How can I see the performance metrics on the test data?
- How can I see all the performance metrics possible for my experiment?
- What if my training/validation and testing data sets come from different distributions?
- Does Driverless AI handle weighted data?
- How does Driverless AI handle fold assignments for weighted data?
- Why do I see that adding new features to a dataset deteriorates the performance of the model?
- Where can I get details of the various transformations performed in an experiment?
- How can I download the predictions onto the machine where Driverless AI is running?
- Why are predicted probabilities not available when I run an experiment without ensembling?
- What drives the size of a MOJO?
- Running the scoring pipeline for my MOJO is taking several hours. How can I get this to run faster?
- Why have I encountered a “Best Score is not finite” error?

Time Series
• What if my data has a time dependency?
• What is a lag, and why does it help?
• Why can’t I specify a validation data set for time-series problems? Why do you look at the test set for time-series problems?
• Why does the gap between train and test matter? Is it because of creating the lag features on the test set?
• In regards to applying the target lags to different subsets of the time group columns, are you saying Driverless AI perform auto-correlation at “levels” of the time series? For example, consider the Walmart dataset where I have Store and Dept (and my target is Weekly Sales). Are you saying that Driverless AI checks for auto-correlation in Weekly Sales based on just Store, just Dept, and both Store and Dept?
• How does Driverless AI detect the time period?
• What is the logic behind the selectable numbers for forecast horizon length?
• Assume that in my Walmart dataset, all stores provided data at the week level, but one store provided data at the day level. What would Driverless AI do?
• Assume that in my Walmart dataset, all stores and departments provided data at the weekly level, but one department in a specific store provided weekly sales on a bi-weekly basis (every two weeks). What would Driverless AI do?
• Why does the number of weeks that you want to start predicting matter?
• Are the scoring components of time series sensitive to the order in which new pieces of data arrive? I.e., is each row independent at scoring time, or is there a real-time windowing effect in the scoring pieces?
• What happens if the user, at predict time, gives a row with a time value that is too small or too large?
• What’s the minimum data size for a time series recipe?
• How long must the training data be compared to the test data?
• How does the time series recipe deal with missing values?
• Can the time information be distributed acrosss multiple columns in the input data (such as [year, day, month])?
• What type of modeling approach does Driverless AI use for time series?
• What’s the idea behind exponential weighting of moving averages?

32.1 General

How is Driverless AI different than any other black box ML algorithm?

Driverless AI uses many techniques (some older and some cutting-edge) for interpreting black box models including creating reason codes for every prediction the system makes. We have also created numerous open source code examples and free publications that explain these techniques. Please see the list below for links to these resources and for references for the interpretability techniques.

• Open source interpretability examples:
  • https://github.com/jphall663/interpretable_machine_learning_with_python
  • https://github.com/h2oai/mli-resources
• Free Machine Learning Interpretability publications:
  • http://www.oreilly.com/data/free/an-introduction-to-machine-learning-interpretablity.csp
• http://docs.h2o.ai/driverless-ai/latest-stable/docs/booklets/MLIBooklet.pdf

• Machine Learning Techniques already in Driverless AI:
  • LOCO: http://www.stat.cmu.edu/~ryantibs/papers/conformal.pdf
  • Surrogate Models:
  • Shapely Explanations: http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions

How often do new versions come out?

The frequency of major new Driverless AI releases has historically been about every two months.

32.2 Installation/Upgrade/Authentication

How can I change my username and password?

The username and password is tied to the experiments you have created. For example, if I log in with the username/password: megan/megan and start an experiment, then I would need to log back in with the same username and password to see those experiments. The username and password, however, does not limit your access to Driverless AI. If you want to use a new user name and password, you can log in again with a new username and password, but keep in mind that you won’t see your old experiments.

Can Driverless AI run on CPU-only machines?

Yes, Driverless AI can run on machines with CPUs only, though GPUs are recommended. Installation instructions are available for GPU and CPU systems. Refer to Installing and Upgrading Driverless AI for more information.

How can I upgrade to a newer version of Driverless AI?

Upgrade instructions vary depending on your environment. Refer to the installation section for your environment. Upgrade instructions are included there.

What kind of authentication is supported in Driverless AI?

Driverless AI supports LDAP, PAM, and Local authentication. These can be configured by setting the appropriate environment variables in the config.toml file or by specifying the environment variables when starting Driverless AI. Refer to Configuring Authentication for more information.

How can I automatically turn on persistence each time the GPU system reboots?

For GPU machines, the sudo nvidia-persistenced --user dai command can be run after each reboot to enable persistence. For systems that have systemd, it is possible to automatically enable persistence after each reboot by removing the --no-persistence-mode flag from nvidia-persistenced.service. Before running the steps below, be sure to review the following for more information:
• https://docs.nvidia.com/deploy/driver-persistence/index.html#persistence-daemon
• https://docs.nvidia.com/deploy/driver-persistence/index.html#installation

1. Run the following to stop the nvidia-persistenced.service:

   ```
   sudo systemctl stop nvidia-persistenced.service
   ```

2. Open the file `/lib/systemd/system/nvidia-persistenced.service`. This file includes a line “ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced --no-persistence-mode --verbose”.

3. Remove the flag `--no-persistence-mode` from that line so that it reads “ExecStart=/usr/bin/nvidia-persistenced --user nvidia-persistenced --verbose”.

4. Run the following command to start the nvidia-persistenced.service:

   ```
   sudo systemctl start nvidia-persistenced.service
   ```

How can I start Driverless AI on a different port that 12345?

**Docker Installs:** When starting Driverless AI in Docker, the `-p` option specifies the port on which Driverless AI will run. Change this option in the start script if you need to run on a port other than 12345. For example, to run on port 443, use the following (change `nvidia-docker run` to `docker-run` if needed):

   ```
   docker run \
   --pid=host \
   --init \n   --rm \
   --shm-size=256m \
   -u `id -u`:`id -g` \
   -p 443:12345 \
   -v `pwd`/data:/data \
   -v `pwd`/log:/log \
   -v `pwd`/license:/license \
   -v `pwd`/tmp:/tmp \
   h2oai/dai-centos7-x86_64:TAG
   ```

**Native Installs:** To run on a port other than 12345, update the port value in the `config.toml` file. For example, edit the following to run Driverless AI on port 443:

   ```
   # Export the Driverless AI config.toml file (or add it to ~/.bashrc)
   export DRIVERLESS_AI_CONFIG_FILE="/config/config.toml"

   # IP address and port for Driverless AI HTTP server.
   ip = "127.0.0.1"
   port = 443
   ```

   Point to this updated config file when restarting Driverless AI.

Can I set up SSL on Driverless AI?

Yes, you can set up HTTPS/SSL on Driverless AI running in an AWS environment. HTTPS/SSL needs to be configured on the host machine, and the necessary ports will need to be opened on the AWS side. You will need to have your own SSL cert or you can create a self signed cert for yourself.

The following is a very simple example showing how to configure HTTPS with a proxy pass to the port on the container 12345 with the keys placed in `/etc/nginx/`. Replace `<server_name>` with your server name.

I received a “Must have exactly one OpenCL platform ‘NVIDIA CUDA’” error. How can I fix that?

If you encounter problems with opencl errors at server time, you may see the following message:

```
2018-11-08 14:26:15,341 C: D:452.2GB M:246.0GB 21603 ERROR : Must have exactly one OpenCL platform 'NVIDIA CUDA', but got:
Platform #0: Clover
Platform #1: NVIDIA CUDA
  -- Device #0: GeForce GTX 1080 Ti
  -- Device #1: GeForce GTX 1080 Ti
  -- Device #2: GeForce GTX 1080 Ti
Please uninstall all but 'NVIDIA CUDA' platform.
```

For Ubuntu, the solution is to run the following:

```
sudo apt-get remove mesa-opencl-icd
```

Is it possible for multiple users to share a single Driverless AI instance?
Driverless AI supports multiple users, and Driverless AI is licensed per a single named user. Therefore, in order, to have different users run experiments simultaneously, they would each need a license. Driverless AI manages the GPU(s) that it is given and ensures that different experiments from different users can run safely simultaneously and don’t interfere with each other. So when two licensed users log in with different credentials, then neither of them will see the other’s experiment. Similarly, if a licensed user logs in using a different set of credentials, then that user will not see any previously run experiments.

**Can multiple Driverless AI users share a GPU server?**

Yes, you can allocate multiple users in a single GPU box. For example, a single box with four GPUs can allocate that User1 has two GPUs and User2 has the other two GPUs. This is accomplished by having two separated Driverless AI instances running on the same server.

There are two ways to assign specific GPUs to Driverless AI. And in the scenario with four GPUs (two GPUs allocated to two users), both of these options allow each Docker container only to see two GPUs.

- **Use the `CUDA_VISIBLE_DEVICES` environment variable.** In the case of Docker deployment, this will translate in passing the `-e CUDA_VISIBLE_DEVICES="0,1"` to the `nvidia-docker run` command.

- **Passing the `NV_GPU` option at the beginning of the `nvidia-docker run` command.** (See example below.)

```bash
# Team 1
NV_GPU='0,1' nvidia-docker run
  --pid=host
  --init
  --rm
  --shm-size=256m
  -u id -u:id -g
  -p port-to-team:12345
  -e DRIVERLESS_AI_CONFIG_FILE="/config/config.toml"
  -v /data:/data
  -v /log:/log
  -v /license:/license
  -v /tmp:/tmp
  -v /config:/config
  h2oai/dai-centos7-x86_64:TAG

# Team 2
NV_GPU='0,1' nvidia-docker run
  --pid=host
  --init
  --rm
  --shm-size=256m
  -u id -u:id -g
  -p port-to-team:12345
  -e DRIVERLESS_AI_CONFIG_FILE="/config/config.toml"
  -v /data:/data
  -v /log:/log
  -v /license:/license
  -v /tmp:/tmp
  -v /config:/config
  h2oai/dai-centos7-x86_64:TAG
```

Note, however, that a Driverless AI instance expects to fully utilize and not share the GPUs that are assigned to it. Sharing a GPU with other Driverless AI instances or other running programs can result in out-of-memory issues.
How can I retrieve a list of Driverless AI users?

A list of users can be retrieved using the Python client.

```python
h2o = Client(address='http://<client_url>:12345', username='<username>',
            password='<password>')
h2o.get_users()
```

Start of Driverless AI fails on the message "Segmentation fault (core dumped)" on Ubuntu 18/RHEL 7.6. How can I fix this?

This problem is caused by the font `NotoColorEmoji.ttf`, which cannot be processed by the Python matplotlib library. A workaround is to disable the font by renaming it. (Do not use fontconfig because it is ignored by matplotlib.) The following will print out the command that should be executed.

```bash
sudo find / -name "NotoColorEmoji.ttf" 2>/dev/null | xargs -I{} echo sudo mv {} {}.backup
```

### 32.3 Data/Experiments/Predictions

**Is there a file size limit for datasets?**

For GBMs, the file size for datasets is limited by the collective CPU or GPU memory on the system, but we continue to make optimizations for getting more data into an experiment, such as using TensorFlow streaming to stream to arbitrarily large datasets.

**How much memory does Driverless AI require in order to run experiments?**

Right now, Driverless AI requires approximately 10x the size of the data in system memory.

**How many columns can Driverless AI handle?**

Driverless AI has been tested on datasets with 10k columns. When running experiments on wide data, Driverless AI automatically checks if it is running out of memory, and if it is, it reduces the number of features until it can fit in memory. This may lead to a worse model, but Driverless AI shouldn’t crash because the data is wide.

**How should I use Driverless AI if I have large data?**

Driverless AI can handle large datasets out of the box. For very large datasets (more than 10 billion rows x columns), we recommend sampling your data for Driverless AI. Keep in mind that the goal of driverless AI is to go through many features and models to find the best modeling pipeline, and not to just train a few models on the raw data (H2O-3 is ideally suited for that case).

For large datasets, the recommended steps are:

1. Run with the recommended accuracy/time/interpretability settings first, especially accuracy <= 7
2. Gradually increase accuracy settings to 7 and choose accuracy 9 or 10 only after observing runs with <= 7.

**How does Driverless AI detect the ID column?**

The ID column logic is one of the following:

- The column is named ‘id’, ‘Id’, ‘ID’ or ‘iD’ exactly
- The column contains a significant number of unique values (above `max_relative_cardinality` in the config.toml file or Max. allowed fraction of uniques for integer and categorical cols in Expert settings)
Can Driverless AI handle data with missing values/nulls?

Yes, data that is imported into Driverless AI can include missing values. Feature engineering is fully aware of missing values, and missing values are treated as information - either as a special categorical level or as a special number. So for target encoding, for example, rows with a certain missing feature will belong to the same group. For Categorical Encoding where aggregations of a numeric columns are calculated for a grouped categorical column, missing values are kept. The formula for calculating the mean is the sum of non-missing values divided by the count of all non-missing values. For clustering, we impute missing values. And for frequency encoding, we count the number of rows that have a certain missing feature.

The imputation strategy is as follows:

- XGBoost/LightGBM do not need missing value imputation and may, in fact, perform worse with any specific other strategy unless the user has a strong understanding of the data.
- Driverless AI automatically imputes missing values using the mean for GLM.
- Driverless AI provides an imputation setting for TensorFlow in the config.toml file: `tf_nan_impute_value post-normalization`. If you set this option to 0, then missing values will be imputed. Setting it to (for example) +5 will specify 5 standard deviations outside the distribution. The default for TensorFlow is -5, which specifies that TensorFlow will treat NAs like a missing value. We recommend that you specify 0 if the mean is better.

How does Driverless AI deal with categorical variables? What if an integer column should really be treated as categorical?

If a column has string values, then Driverless AI will treat it as a categorical feature. There are multiple methods for how Driverless AI converts the categorical variables to numeric. These include:

- One Hot Encoding: creating dummy variables for each value
- Frequency Encoding: replace category with how frequently it is seen in the data
- Target Encoding: replace category with the average target value (additional steps included to prevent overfitting)

Driverless AI will try multiple methods for representing the column and determine which representation(s) are best.

If the column has integers, Driverless AI will try treating the column as a categorical column and numeric column. It will treat any integer column as both categorical and numeric if the number of unique values is less than 50.

This is configurable in the config.toml file:

```toml
# Whether to treat some numerical features as categorical
# For instance, sometimes an integer column may not represent a numerical feature but represent different numerical codes instead.
num_as_cat = true

# Max number of unique values for integer/real columns to be treated as categorical (test applies to first statistical_threshold_data_size_small rows only)
max_int_as_cat_uniques = 50
```
(Note: Driverless AI will also check if the distribution of any numeric column differs significantly from the distribution of typical numerical data using Benford’s Law. If the column distribution does not obey Benford’s Law, we will also try to treat it as categorical even if there are more than 50 unique values.)

How are outliers handled?

Outliers are not removed from the data. Instead Driverless AI finds the best way to represent data with outliers. For example, Driverless AI may find that binning a variable with outliers improves performance.

For target columns, Driverless AI first determines the best representation of the column. It may find that for a target column with outliers, it is best to predict the log of the column.

If I drop several columns from the Train dataset, will Driverless AI understand that it needs to drop the same columns from the Test dataset?

If you drop columns from the training dataset, Driverless AI will do the same for the validation and test datasets (if the columns are present). There is no need for these columns because no features will be created from them.

Does Driverless AI treat numeric variables as categorical variables?

In certain cases, yes. You can prevent this behavior by setting the `num_as_cat` variable in your installation’s `config.toml` file to `false`. You can have finer grain control over this behavior by excluding the Numeric to Categorical Target Encoding Transformer and the Numeric To Categorical Weight of Evidence Transformer and their corresponding genes in your installation’s `config.toml` file. To learn more about the `config.toml` file, see the Using the `config.toml` File section.

Which algorithms are used in Driverless AI?

Features are engineered with a proprietary stack of Kaggle-winning statistical approaches including some of the most sophisticated target encoding and likelihood estimates based on groupings, aggregations and joins, but we also employ linear models, neural nets, clustering and dimensionality reduction models and many traditional approaches such as one-hot encoding etc.

On top of the engineered features, sophisticated models are fitted, including, but not limited to: XGBoost (both original XGBoost and ‘lossguide’ (LightGBM) mode), GLM, TensorFlow (including a TensorFlow NLP recipe based on CNN Deeplearning models), RuleFit, and FTRL (Follow the Regularized Leader). More will continue to be added later.

In general, GBMs are the best single-shot algorithms. Since 2006, boosting methods have proven to be the most accurate for noisy predictive modeling tasks outside of pattern recognition in images and sound (https://www.cs.cornell.edu/~caruana/ctp/ct.papers/caruana.icml06.pdf). The advent of XGBoost and Kaggle only cemented this position.

How can we turn on TensorFlow Neural Networks so they are evaluated?

Neural networks are considered by Driverless AI, although they may not be evaluated by default. To ensure that neural networks are tried, you can turn on TensorFlow in the Expert Settings:
Once you have set TensorFlow to ON. You should see the Experiment Preview on the left hand side change and mention that it will evaluate TensorFlow models:
We recommend using TensorFlow neural networks if you have a multinomial use case with more than 5 unique values.

**Does Driverless AI standardize the data?**

Driverless AI will automatically do variable standardization for certain algorithms. For example, with Linear Models and Neural Networks, the data is automatically standardized. For decision tree algorithms, however, we do not perform standardization since these algorithms do not benefit from standardization.

**What objective function is used in XGBoost?**

The objective function used in XGBoost is:

- reg:linear and custom absolute error objective function for regression
- binary:logistic or multi:softprob for classification

The objective function does not change depending on the scorer chosen. The scorer influences parameter tuning only.

For regression, Tweedie/Gamma/Poisson/etc. regression is not yet supported, but Driverless AI handles various target transforms so many target distributions can be handled very effectively already. Driverless AI handles quantile regression for alpha=0.5 (media), and general quantiles are on the roadmap.

Further details for the XGBoost instantiations can be found in the logs and in the model summary, both of which can be downloaded from the GUI or are found in the `/tmp/h2oai_experiment_<name>/` folder on the server.

**Does Driverless AI perform internal or external validation?**

Driverless AI does internal validation when only training data is provided. It does external validation when training and validation data are provided. In either scenario, the validation data is used for all parameter tuning (models and features), not just for feature selection. Parameter tuning includes target transformation, model selection, feature engineering, feature selection, stacking, etc.

Specifically:

- **Internal validation (only training data given):**
  - Ideal when data is either close to i.i.d., or for time-series problems
  - Internal holdouts are used for parameter tuning, with temporal causality for time-series problems
  - Will do the full spectrum from single holdout split to 5-fold CV, depending on accuracy settings
  - No need to split training data manually
  - Final models are trained using CV on the training data

- **External validation (training + validation data given):**
  - Ideal when there’s some amount of drift in the data, and the validation set mimics the test set data better than the training data
  - No training data wasted during training because training data not used for parameter tuning
  - Validation data is used only for parameter tuning, and is not part of training data
  - No CV possible because we explicitly do not want to overfit on the training data
  - Not allowed for time-series problems (see Time Series FAQ section that follows)
**Tip:** If you want both training and validation data to be used for parameter tuning (the training process), just concatenate the datasets together and turn them both into training data for the “internal validation” method.

**How does Driverless AI prevent overfitting?**

Driverless AI performs a number of checks to prevent overfitting. For example, during certain transformations, Driverless AI calculates the average on out-of-fold data using cross validation. Driverless AI also performs early stopping for every model built, ensuring that the model build will stop when it ceases to improve on holdout data. And additional steps to prevent overfitting include checking for i.i.d. and avoiding leakage during feature engineering.

A blog post describing Driverless AI overfitting protection in greater detail is available here: [https://www.h2o.ai/blog/driverless-ai-prevents-overfitting-leakage/](https://www.h2o.ai/blog/driverless-ai-prevents-overfitting-leakage/).

**How does Driverless AI avoid the multiple hypothesis (MH) problem?**

Or more specifically, like many brute force methods for tuning hyperparameters/model selection, Driverless AI runs up against the multihypothesis problem (MH). For example, if I randomly generated a gazillion models, the odds that a few will do awesome on the test data that they are all measured against is pretty large, simply by sheer luck. How does Driverless AI address this?

Driverless AI uses a variant of the reusable holdout technique to address the multiple hypothesis problem. Refer to [https://pdfs.semanticscholar.org/25fe/96591144f4af3d8f8f79ec95b37f415e5bb75.pdf](https://pdfs.semanticscholar.org/25fe/96591144f4af3d8f8f79ec95b37f415e5bb75.pdf) for more information.

**How does Driverless AI suggest the experiment settings?**

When you run an experiment on a dataset, the experiment settings (Accuracy, Time, and Interpretability) are automatically suggested by Driverless AI. For example, Driverless AI may suggest the parameters Accuracy = 7, Time = 3, Interpretability = 6, based on your data.

Driverless AI will automatically suggest experiment settings based on the number of columns and number of rows in your dataset. The settings are suggested to ensure best handling when the data is small. If the data is small, Driverless AI will suggest the settings that prevent overfitting and ensure the full dataset is utilized.

If the number of rows and number of columns are each below a certain threshold, then:

- Accuracy will be increased up to 8.
  - The accuracy is increased so that cross validation is done. (We don’t want to “throw away” any data for internal validation purposes.)

- Interpretability will be increased up to 8.
  - The higher the interpretability setting, the smaller the number of features in the final model.
  - More complex features are not allowed.
  - This prevents overfitting.

- Time will be decreased down to 2.
  - There will be fewer feature engineering iterations to prevent overfitting.

**What happens when I set Interpretability and Accuracy to the same number?**

The answer is currently that interpretability controls which features are created and what features are kept. (Also above interpretability = 6, monotonicity constraints are used in XGBoost.) The accuracy refers to how hard Driverless AI then tries to make those features into the most accurate model.

**Can I specify the number of GPUs to use when running Driverless AI?**
When running an experiment, the Expert Settings allow you to specify the starting GPU ID for Driverless AI to use. You can also specify the maximum number of GPUs to use per model and per experiment. Refer to the *Expert Settings* section for more information.

**How can I create the simplest model in Driverless AI?**

To create the simplest model in Driverless AI, set the following Experiment Settings:

- Set Accuracy to 1. Note that this can hurt performance as a sample will be used. If necessary, adjust the knob until the preview shows no sampling.
- Set Time to 1.
- Set Interpretability to 10.

Next, configure the following Expert Settings:

- Turn OFF all algorithms except GLM.
- Set GLM models to ON.
- Set Ensemble level to 0.
- Set `Select target transformation of the target for regression problems` to Identity.
- Disable Data distribution shift detection.
- Disable Target Encoding.

Alternatively, you can set Pipeline Building Recipe to Compliant. Compliant automatically configures the following experiment and expert settings:

- interpretability=10 (To avoid complexity. This overrides GUI or Python client settings for Interpretability.)
- enable_glm=’on’ (Remaining algos are ‘off’, to avoid complexity and be compatible with algorithms supported by MLI.)
- num_as_cat=false: Don’t convert any numerics to categoricals except via one-hot encoding (to avoid complexity).
- fixed_ensemble_level=0: Don’t use any ensemble (to avoid complexity).
- feature_brain_level=0: No feature brain used (to ensure every restart is identical).
- max_feature_interaction_depth=1: Interaction depth is set to 1 (no multi-feature interactions to avoid complexity).
- target_transformer=’identity’: For regression (to avoid complexity).
- check_distribution_shift=false: Don’t use distribution shift between train, valid, and test to drop features (bit risky without fine-tuning).

**Why is my experiment suddenly slow?**

It is possible that your experiment has gone from using GPUs to using CPUs due to a change of the host system outside of Driverless AI’s control. You can verify this using any of the following methods:

- Check GPU usage by going to your Driverless AI experiment page and clicking on the GPU USAGE tab in the lower-right quadrant of the experiment.
- Run `nvidia-smi` in a terminal to see if any processes are using GPU resources in an unexpected way (such as those using a large amount of memory).
- Check if System/GPU memory is being consumed by prior jobs or other tasks or if older jobs are still running some tasks.
• Check and disable automatic NVIDIA driver updates on your system (as they can interfere with running experiments).

The general solution to these kind of sudden slowdown problems is to restart:
• Restart Docker if using Docker
• `pkill --signal 9 h2oai` if using the native installation method
• Restart the system if `nvidia-smi` does not work as expected (e.g., after a driver update)

More ML-related issues that can lead to a slow experiment are:
• Choosing high accuracy settings on a system with insufficient memory
• Choosing low interpretability settings (can lead to more feature engineering which can increase memory usage)
• Using a dataset with a lot of columns (> 500)
• Doing multi-class classification with a GBM model when there are many target classes (> 5)

When I run multiple experiments with different seeds, why do I see different scores, runtimes, and sizes on disk in the Experiments listing page?

When running multiple experiments with all of the same settings except the seed, understand that a feature brain level > 0 can lead to variations in models, features, timing, and sizes on disk. (The default value is 2.) These variations can be disabled by setting the Feature Brain Level to 0 in the Expert Settings or in the config.toml file.

In addition, if you use a different seed for each experiment, then each experiment can be different due to the randomness in the genetic algorithm that searches for the best features and model parameters. Only if Reproducible is set with the same seed and with the a feature brain level of 0 should users expect the same outcome. Once a different seed is set, the models, features, timing, and sizes on disk can all vary within the constraints set by the choices made for the experiment. (I.e., accuracy, time, interpretability, expert settings, etc., all constrain the outcome, and then a different seed can change things within those constraints.)

Why does the final model performance appear to be worse than previous iterations?

There are a few things to remember:
• Driverless AI creates a best effort estimate of the generalization performance of the best modeling pipeline found so far
• The performance estimation is always based on holdout data (data unseen by the model).
• If no validation dataset is provided, the training data is split internally to create internal validation holdout data (once or multiple times or cross-validation, depending on the accuracy settings).
• If no validation dataset is provided, for accuracy <= 7, a single holdout split is used, and a “lucky” or “unlucky” split can bias estimates for small datasets or datasets with high variance.
• If a validation dataset is provided, then all performance estimates are solely based on the entire validation dataset (independent of accuracy settings).
• All scores reported are based on bootstrapped-based statistical methods and come with error bars that represent a range of estimate uncertainty.

After the final iteration, a best final model is trained on a final set of engineered features. Depending on accuracy settings, a more accurate estimation of generalization performance may be done using cross-validation. Also, the final model may be a stacked ensemble consisting of multiple base models, which generally leads to better performance. Consequently, in rare cases, the difference in performance estimation method can lead to the final model’s estimated performance seeming poorer than those from previous iterations.
iterations. (i.e., The final model’s estimated score is significantly worse than the last iteration score and error bars don’t overlap.) In that case, it is very likely that the final model performance estimation is more accurate, and the prior estimates were biased due to a “lucky” split. To confirm this, you can re-run the experiment multiple times (without setting the reproducible flag).

If you would like to minimize the likelihood of the final model performance appearing worse than previous iterations, here are some recommendations:

- Increase accuracy settings
- Provide a validation dataset
- Provide more data

**How can I find features that may be causing data leakages in my Driverless AI model?**

To find original features that are causing leakage, have a look at features_orig.txt in the experiment summary download. Features causing leakage will have high importance there. To get a hint at derived features that might be causing leakage, create a new experiment with dials set to 2/2/8, and run the new experiment on your data with all your features and response. Then analyze the top 1-2 features in the model variable importance. They are likely the main contributors to data leakage if it is occurring.

**How can I see the performance metrics on the test data?**

As long as you provide a target column in the test set, Driverless AI will show the best estimate of the final model’s performance on the test set at the end of the experiment. The test set is never used to tune parameters (unlike to what Kagglers often do), so this is purely a convenience. Of course, you can still make test set predictions and compute your own metrics using a method of your choice.

**How can I see all the performance metrics possible for my experiment?**

At the end of the experiment, the model’s estimated performance on all provided datasets with a target column is printed in the experiment logs. For example, for the test set:

```
Final scores on test (external holdout) +/- stddev:
  GINI = 0.87794 +/- 0.035305 (more is better)
  MCC = 0.71124 +/- 0.043232 (more is better)
  F05 = 0.79175 +/- 0.04209 (more is better)
  F1 = 0.75823 +/- 0.038675 (more is better)
  F2 = 0.82752 +/- 0.03604 (more is better)
  ACCURACY = 0.91513 +/- 0.011975 (more is better)
  LOGLOSS = 0.28429 +/- 0.016682 (less is better)
  AUCPR = 0.79074 +/- 0.046223 (more is better)
optimized: AUC = 0.93386 +/- 0.018856 (more is better)
```

**What if my training/validation and testing data sets come from different distributions?**

In general, Driverless AI uses training data to engineer features and train models and validation data to tune all parameters. If no external validation data is given, the training data is used to create internal holdouts. The way holdouts are created internally depends on whether there is a strong time dependence, see the point below. If the data has no obvious time dependency (e.g., if there is no time column neither implicit or explicit), or if the data can be sorted arbitrarily and it won’t affect the outcome (e.g., Iris data, predicting flower species from measurements), and if the test dataset is different (e.g., new flowers or only large flowers), then the model performance on validation (either internal or external) as measured during training won’t be achieved during final testing due to the obvious inability of the model to generalize.

**Does Driverless AI handle weighted data?**

Yes. You can optionally provide an extra weight column in your training (and validation) data with non-negative observation weights. This can be useful to implement domain-specific effects such as exponential
weighting in time or class weights. All of our algorithms and metrics in Driverless AI support observation weights, but note that estimated likelihoods can be skewed as a consequence.

**How does Driverless AI handle fold assignments for weighted data?**

Currently, Driverless AI does not take the weights into account during fold creation, but you can provide a fold column to enforce your own grouping, i.e., to keep rows that belong to the same group together (either in train or valid). The fold column has to be a categorical column (integers ok) that assigns a group ID to each row. (It needs to have at least 5 groups because we do up to 5-fold CV.)

**Why do I see that adding new features to a dataset deteriorates the performance of the model?**

You may notice that after adding one or more new features to a dataset, it deteriorates the performance of the Driverless AI model. In Driverless AI, the feature engineering sequence is fairly random and may end up not doing same things with original features if you restart entirely fresh with new columns.

Beginning in Driverless AI v1.4.0, you now have the option to **Restart from Last Checkpoint**. This allows you to pull in a new dataset with more columns, and Driverless AI will more iteratively take advantage of the new columns.

**Where can I get details of the various transformations performed in an experiment?**

Download the experiment’s log .zip file from the GUI. This zip file includes summary information, log information, and a gene_summary.txt file with details of the transformations used in the experiment. Specifically, there is a **details** folder with all subprocess logs.

On the server, the experiment specific files are inside the `/tmp/h2oai_experiment_<name>/` folder after the experiment completes, particularly `h2oai_experiment_logs_<name>.zip` and `h2oai_experiment_summary_<name>.zip`.

**How can I download the predictions onto the machine where Driverless AI is running?**

When you select **Score on Another Dataset**, the predictions will automatically be stored on the machine where Driverless AI is running. They will be saved in the following locations (and can be opened again by Driverless AI, both for .csv and .bin):

- **Training Data Predictions**: `tmp/h2oai_experiment_<name>/train_preds.csv` (also saved as .bin)
- **Testing Data Predictions**: `tmp/h2oai_experiment_<name>/test_preds.csv` (also saved as .bin)
- **New Data Predictions**: `tmp/h2oai_experiment_<name>/automatically_generated_name.csv` Note that the automatically generated name will match the name of the file downloaded to your local computer.

**Why are predicted probabilities not available when I run an experiment without ensembling?**

When Driverless AI provides pre-computed predictions after completing an experiment, it uses only those parts of the modeling pipeline that were not trained on the particular rows for which the predictions are made. This means that Driverless AI needs holdout data in order to create predictions, such as validation or test sets, where the model is trained on training data only. In the case of ensembles, Driverless AI uses cross-validation to generate holdout folds on the training data, so we are able to provide out-of-fold estimates for every row in the training data and, hence, can also provide training holdout predictions (that will provide a good estimate of generalization performance). In the case of a single model, though, that is trained on 100% of the training data. There is no way to create unbiased estimates for any row in the training data. While DAI uses an internal validation dataset, this is a re-usable holdout, and therefore will not contain holdout predictions for the full training dataset. You need cross-validation in order to get out-of-fold estimates, and then that’s not a single model anymore. If you want to still get predictions for the training data for a single model, then you have to use the scoring API to create predictions on the training set. From the GUI, this can be done using the **Score on Another Dataset** button for a completed experiment. Note, though, that the results will likely be overly optimistic, too good to be true, and virtually useless.
What drives the size of a MOJO?

The size of the MOJO is based on the complexity of the final modeling pipeline (i.e., feature engineering and models). One of the biggest factors is the amount of higher-order interactions between features, especially target encoding and related features, which have to store lookup tables for all possible combinations observed in the training data. You can reduce the amount of these transformations by reducing the value of **Max. feature interaction depth** and/or **Feature engineering effort** under Expert Settings, or by increasing the interpretability settings for the experiment. Ensembles also contribute to the final modeling pipeline’s complexity as each model has its own pipeline. Lowering the accuracy settings or increasing the **ensemble_accuracy_switch** setting in the config.toml file can help here. The number of features **Max. pipeline features** also affects the MOJO size. Text transformers are pretty bulky as well and can add to the MOJO size.

Running the scoring pipeline for my MOJO is taking several hours. How can I get this to run faster?

When running example.sh, Driverless AI implements a memory setting, which is suitable for most use cases. For very large models, however, it may be necessary to increase the memory limit when running the Java application for data transformation. This can be done using the `-Xmx25g` parameter. For example:

```
java -Xmx25g -Dai.h2o.mojos.runtime.license.file=license.sig -cp mojo2-˓
  →runtime.jar ai.h2o.mojos.ExecuteMojo pipeline.mojo example.csv
```

Why have I encountered a “Best Score is not finite” error?

Driverless AI uses 32-bit floats by default. You may encounter this error if your data value exceeds 1E38 or if you are resolving more than 1 part in 10 million. You can resolve this error using one of the following methods:

- Enable the **Force 64-bit Precision** option in the experiment’s Expert Settings.

or

- Set `data_precision="float64"` and `transformer_precision="float64"` in config.toml.

32.4 Time Series

What if my data has a time dependency?

If you know that your data has a strong time dependency, select a time column before starting the experiment. The time column must be in a Datetime format that can be parsed by pandas, such as “2017-11-06 14:32:21”, “Monday, June 18, 2012” or “Jun 18 2018 14:34:00” etc., or contain only integers.

If you are unsure about the strength of the time dependency, run two experiments: One with time column set to “[OFF]” and one with time column set to “[AUTO]” (or pick a time column yourself).

What is a lag, and why does it help?

A lag is a feature value from a previous point in time. Lags are useful to take advantage of the fact that the current (unknown) target value is often correlated with previous (known) target values. Hence, they can better capture target patterns along the time axis.

Why can’t I specify a validation data set for time-series problems? Why do you look at the test set for time-series problems

The problem with validation vs test in the time series setting is that there is only one valid way to define the split. If a test set is given, its length in time defines the validation split and the validation data has to be part of train. Otherwise the time-series validation won’t be useful.
For instance: Let’s assume we have train = [1,2,3,4,5,6,7,8,9,10] and test = [12,13], where integers define time periods (e.g., weeks). For this example, the most natural train/valid split that mimics the test scenario would be: train = [1,2,3,4,5,6,7] and valid = [9,10], and month 8 is not included in the training set to allow for a gap. Note that we will look at the start time and the duration of the test set only (if provided), and not at the contents of the test data (neither features nor target). If the user provides validation = [8,9,10] instead of test data, then this could lead to inferior validation strategy and worse generalization. Hence, we use the user-given test set only to create the optimal internal train/validation splits. If no test set is provided, the user can provide the length of the test set (in periods), the length of the train/test gap (in periods) and the length of the period itself (in seconds).

Why does the gap between train and test matter? Is it because of creating the lag features on the test set?

Taking the gap into account is necessary in order to avoid too optimistic estimates of the true error and to avoid creating history-based features like lags for the training and validation data (which cannot be created for the test data due to the missing information).

In regards to applying the target lags to different subsets of the time group columns, are you saying Driverless AI perform auto-correlation at “levels” of the time series? For example, consider the Walmart dataset where I have Store and Dept (and my target is Weekly Sales). Are you saying that Driverless AI checks for auto-correlation in Weekly Sales based on just Store, just Dept, and both Store and Dept?

Currently, auto-correlation is only applied on the detected superkey (entire TGC) of the training dataset relation at the very beginning. It’s used to rank potential lag-sizes, with the goal to prune the search space for the GA optimization process, which is responsible for selecting the lag features.

How does Driverless AI detect the time period?

Driverless AI treats each time series as a function with some frequency 1/ns. The actual value is estimated by the median of time deltas across maximal length TGC subgroups. The chosen SI unit minimizes the distance to all available SI units.

What is the logic behind the selectable numbers for forecast horizon length?

The shown forecast horizon options are based on quantiles of valid splits. This is necessary because Driverless AI cannot display all possible options in general.

Assume that in my Walmart dataset, all stores provided data at the week level, but one store provided data at the day level. What would Driverless AI do?

Driverless AI would still assume “weekly data” in this case because the majority of stores are yielding this property. The “daily” store would be resampled to the detected overall frequency.

Assume that in my Walmart dataset, all stores and departments provided data at the weekly level, but one department in a specific store provided weekly sales on a bi-weekly basis (every two weeks). What would Driverless AI do?

That’s similar to having missing data. Due to proper resampling, Driverless AI can handle this without any issues.

Why does the number of weeks that you want to start predicting matter?

That’s an option to provide a train-test gap if there is no test data is available. That is to say, “I don’t have my test data yet, but I know it will have a gap to train of x.”

Are the scoring components of time series sensitive to the order in which new pieces of data arrive? i.e., is each row independent at scoring time, or is there a real-time windowing effect in the scoring pieces?

Each row is independent at scoring time.

What happens if the user, at predict time, gives a row with a time value that is too small or too large?

Internally, “out-of-bounds” time values are encoded with special values. The samples will still be scored, but the predictions won’t be trustworthy.
What’s the minimum data size for a time series recipe?

We recommended that you have around 10,000 validation samples in order to get a reliable estimate of the true error. The time series recipe can still be applied for smaller data, but the validation error might be inaccurate.

How long must the training data be compared to the test data?

At a minimum, the training data has to be at least twice as long as the test data along the time axis. However, we recommended that the training data is at least three times as long as the test data.

How does the time series recipe deal with missing values?

Missing values will be converted to a special value, which is different from any non-missing feature value. Explicit imputation techniques won’t be applied.

Can the time information be distributed across multiple columns in the input data (such as [year, day, month])?

Currently Driverless AI requires the data to have the time stamps given in a single column. Driverless AI will create addition time features like [year, day, month] on its own, if they turn out to be useful.

What type of modeling approach does Driverless AI use for time series?

Driverless AI combines the creation of history-based features like lags, moving averages etc. with the modeling techniques, which are also applied for i.i.d. data. The primary model of choice is XGBoost.

What’s the idea behind exponential weighting of moving averages?

Exponential weighting accounts for the possibility that more recent observations are better suited to explain the present than older observations.
This section includes Arno’s tips for running Driverless AI.

### 33.1 Time Series Tips

H2O Driverless AI handles time-series forecasting problems out of the box.

All you need to do when starting a time-series experiment is to provide a regular columnar dataset containing your features. Then pick a target column and also pick a “time column” - a designated column containing time stamps for every record (row) such as “April 10 2019 09:13:41” or “2019/04/10”. If you have a test set for which you want predictions for every record, make sure to provide future time stamps and features as well.

In most cases, that’s it. You can launch the experiment and let Driverless AI do the rest. It will even auto-detect multiple time series in the same dataset for different groups such as weekly sales for stores and departments (by finding the columns that identify stores and departments to group by). Driverless AI will also auto-detect the time period including potential gaps during weekends, as well as the forecast horizon, a possible time gap between training and testing time periods (to optimize for deployment delay) and even keeps track of holiday calendars. Of course, it automatically creates multiple causal time-based validation splits (sliding time windows) for proper validation, and incorporates many other related grand-master recipes such as automatic target and non-target lag feature generation as well as interactions between lags, first and second derivatives and exponential smoothing.

- If you find that the automatic lag-based time-series recipe isn’t performing well for your dataset, we recommend that you try to disable the creation of lag-based features by disabling “Time-series lag-based recipe” in the expert settings. This will lead to regular feature engineering but with time-based causal validation splits. Especially for small datasets and short forecast periods, this can lead to better results.

- If the target column is present in the test set and has partially filled information (non-missing values), then Driverless AI will automatically augment the model with those future target values to make better predictions. This can be used to extend the usable lifetime of the model into the future without the need for retraining by providing past known outcomes. Contact us if you’re interested in learning more about test-time augmentation.

- For now, training and test datasets should have the same input features available, so think about which of the predictors (input features) will be available during production time and drop the rest (or create your own lag features that can be available to both train and test sets).

- For datasets that are non-stationary in time, create a test set from the last temporal portion of data, and create time-based features. This allows the model to be optimized for your production scenario.

- We are working on further improving many aspects of our time-series recipe. For example, we will add support to automatically generate lags for features that are only available in the training set, but not in the test set, such as environmental or economic factors. We’ll also improve the performance of back-testing using rolling windows.
• In 1.7.x, you will have the option to bring your own recipes (BYOR) for features, models and scorers, and that includes time-series recipes! We are very excited about that. Please contact us if you are interested in learning more about BYOR.

### 33.2 Scorer Tips

A core capability of H2O Driverless AI is the creation of automatic machine learning modeling pipelines for supervised problems. In addition to the data and the target column to be predicted, the user can pick a scorer. A scorer is a function that takes actual and predicted values for a dataset and returns a number. Looking at this single number is the most common way to estimate the generalization performance of a predictive model on unseen data by comparing the model’s predictions on the dataset with its actual values. There are more detailed ways to estimate the performance of a machine learning model such as residual plots (available on the Diagnostics page in Driverless AI), but we will focus on scorers here.

For a given scorer, Driverless AI optimizes the pipeline to end up with the best possible score for this scorer. The default scorer for regression problems is RMSE (root mean squared error), where 0 is the best possible value. For example, for a dataset containing 4 rows, if actual target values are [1, 1, 10, 0], but predictions are [2, 3, 4, -1], then the RMSE is $\sqrt{(1+4+36+1)/4}$ and the largest misprediction dominates the overall score (quadratically). Driverless AI will focus on improving the predictions for the third data point, which can be very difficult when hard-to-predict outliers are present in the data. If outliers are not that important to get right, a metric like the MAE (mean absolute error) can lead to better results. For this case, the MAE is $(1+2+6+1)/4$ and the optimization process will consider all errors equally (linearly). Another scorer that is robust to outliers is RMSLE (root mean square logarithmic error), which is like RMSE but after taking the logarithm of actual and predicted values - however, it is restricted to positive values. For price predictions, scorers such as MAPE (mean absolute percentage error) or MER (median absolute percentage error) are useful, but have problems with zero or small positive values. SMAPE (symmetric mean absolute percentage error) is designed to improve upon that.

For classification problems, the default scorer is either the AUC (area under the receiver operating characteristic curve) or LOGLOSS (logarithmic loss) for imbalanced problems. LOGLOSS focuses on getting the probabilities right (strongly penalizes wrong probabilities), while AUC is designed for ranking problems. Gini is similar to the AUC, but measures the quality of ranking (inequality) for regression problems. For general imbalanced classification problems, AUCPR and MCC are good choices, while F05, F1 and F2 are designed to balance recall against precision.

We highly suggest experimenting with different scorers and to study their impact on the resulting models. Using the Diagnostics page in Driverless AI, all applicable scores can be computed for any given model, no matter which scorer was used during training.

### 33.3 Knob Settings Tips

H2O Driverless AI allows you to customize every experiment in great detail via the expert settings. The most important controls however are the three knobs for accuracy, time and interpretability. A higher accuracy setting results in a better estimate of the model generalization performance, usually through using more data, more holdout sets, more parameter tuning rounds and other advanced techniques. Higher time settings means the experiment is given more time to converge to an optimal solution. Higher interpretability settings reduces the model’s complexity through less feature engineering and using simpler models. In general, a setting of 1/1/10 will lead to the simplest and usually least accurate modeling pipeline, while a setting of 10/10/1 will lead to the most complex and most time consuming experiment possible. Generally, it is sufficient to use settings of 7/5/5 or similar, and we recommend to start with the default settings. We highly recommend studying the experiment preview on the left-hand side of the GUI before each experiment - it can help you fine-tune the settings and save time overall.

Note that you can always finish an experiment early, either by clicking ‘Finish’ to get the deployable final pipeline out, or by clicking ‘Abort’ to instantly terminate the experiment. In either case, the experiment can be continued seamlessly.
at a later time with ‘Restart from last Checkpoint’ or ‘Retrain Final Pipeline’, and you can always turn the knobs (or modify the expert settings) to adapt to your requirements.

### 33.4 Tips for Running an Experiment

H2O Driverless AI is an automatic machine learning platform designed to create highly accurate modeling pipelines from tabular training data. The predictive performance of the pipeline is a function of both the training data and the parameters of the pipeline (details of feature engineering and modeling). During an experiment, Driverless AI automatically tunes these parameters by scoring candidate pipelines on held out (“validation”) data. This important validation data is either provided by the user (for experts) or automatically created (random, time-based or fold-based) by Driverless AI. Once a final pipeline has been created, it should be scored on yet another held out dataset (“test data”) to estimate its generalization performance. Understanding the origin of the training, validation and test datasets (“the validation scheme”) is critical for success with machine learning, and we welcome your feedback and suggestions to help us create the right validation schemes for your use cases.

### 33.5 Expert Settings Tips

H2O Driverless AI offers a range of ‘Expert Settings’ that allow you to customize each experiment. For example, you can limit the amount of feature engineering by reducing the value for ‘Feature engineering effort’ or ‘Max. feature interaction depth’ or by disabling ‘Target Encoding’. You can also select the model types to be used for training on the engineered features (such as XGBoost, LightGBM, GLM, TensorFlow, FTRL, or RuleFit). For time-series problems where the selected time_column leads to an error message (this can currently happen if the the time structure is not regular enough - we are working on an improved version), you can disable the ‘Time-series lag-based recipe’ and Driverless AI will create train/validation splits based on the time order instead, which can increase the model’s performance if the time column is important.

### 33.6 Checkpointing Tips

Driverless AI provides the option to checkpoint experiments to speed up feature engineering and model tuning when running multiple experiments on the same dataset. By default, H2O Driverless AI automatically scans all prior experiments (including aborted ones) for an optimal checkpoint to restart from. You can select a specific prior experiment to restart a new experiment from with “Restart from Last Checkpoint” in the experiment listing page (click on the 3 yellow bars on the right). You can disable checkpointing by setting ‘Feature Brain Level’ in the expert settings (or feature_brain_level in the configuration file) to 0 to force the experiment to start from scratch.

### 33.7 Text Data Tips

For datasets that contain text (string) columns - where each value can be a few words, a paragraph or an entire document - Driverless AI automatically creates NLP features based on bag of words, tf-idf, singular value decomposition and out-of-fold likelihood estimates. In versions 1.3 and above, you can enable TensorFlow in the expert settings to see how CNN (convolutional neural net) based learned word embeddings can improve predictive accuracy even more. Try this for sentiment analysis, document classification, and generic text-enriched datasets.
This appendix describes how to install the Driverless AI Python Client. It also provides some end-to-end examples showing how to use the Driverless AI Python client. Additional examples are available in the https://github.com/h2oai/driverlessai-tutorials repository.

34.1 Installing the Python Client

The Python Client is available on the Driverless AI UI and published on the h2oai channel at https://anaconda.org/h2oai/repo.

34.1.1 Installing from Driverless AI

Requirements

- Python 3.6. This is the only supported version.

Download from UI

On the Driverless AI top menu, select the RESOURCES > PY_CLIENT link. This downloads the h2oai_client wheel.

Download from command line

The Driverless AI Python client is exposed as /clients/py HTTP end point. This can be accessed via command line:

```
wget --trust-server-names http://<Driverless AI address>/clients/py
```

Wheel installation

Install this wheel to your local Python via pip command. Once installed, you can launch a Jupyter notebook and begin using the Driverless AI Python Client.
34.1.2 Installing from Anaconda Cloud

Requirements

• Conda Package Manager. You can install the Conda Package Manager either through Anaconda or Miniconda. Note that the Driverless AI Python client requires Python 3.6, so ensure that you install the Python 3 version of Anaconda or Miniconda.
  – Anaconda Install Instructions
  – Miniconda Install Instructions

Installation Procedure

After Conda is installed and the Conda executable is available in $PATH, create a new Anaconda environment for h2oai_client:

```
conda create -n h2oaiclientenv -c h2oai -c conda-forge h2oai_client
```

Upon completion, you can launch a Jupyter notebook and begin using the Driverless AI Python Client.

34.2 Credit Card Demo

This notebook provides an H2OAI Client workflow, of model building and scoring, that parallels the Driverless AI workflow.

Notes:

• This is an early release of the Driverless AI Python client.
• Python 3.6 is the only supported version.
• You must install the h2oai_client wheel to your local Python. This is available from the RESOURCES link in the top menu of the UI.

![Fig. 1: py-client](image)

Fig. 1: py-client
34.2.1 Workflow Steps

Build an Experiment with Python API:

1. Sign in
2. Import train & test set/new data
3. Specify experiment parameters
4. Launch Experiment
5. Examine Experiment
6. Download Predictions

Build an Experiment in Web UI and Access Through Python:

1. Get pointer to experiment

Score on New Data:

1. Score on new data with H2OAI model

Model Diagnostics on New Data:

1. Run model diagnostics on new data with H2OAI model

Run Model Interpretation

1. Run model interpretation on the raw features
2. Run Model Interpretation on External Model Predictions

Build Scoring Pipelines

1. Build Python Scoring Pipeline
2. Build MOJO Scoring Pipeline

34.2.2 Build an Experiment with Python API

1. Sign In

Import the required modules and log in.

Pass in your credentials through the Client class which creates an authentication token to send to the Driverless AI Server. In plain English: to sign into the Driverless AI web page (which then sends requests to the Driverless Server), instantiate the Client class with your Driverless AI address and login credentials.

In [1]:
   import h2oai_client
   import numpy as np
   import pandas as pd
   # import h2o
   import requests
   import math
   from h2oai_client import Client, ModelParameters, InterpretParameters

In [2]:
   address = 'http://ip_where_driverless_is_running:12345'
   username = 'username'
   password = 'password'
   h2oai = Client(address = address, username = username, password = password)
   # make sure to use the same user name and password when signing in through the GUI

34.2. Credit Card Demo
Equivalent Steps in Driverless: Signing In

Fig. 2: Equivalent Steps in Driverless: Signing In

Fig. 3: Equivalent Steps in Driverless: Signing In

2. Upload Datasets

Upload training and testing datasets from the Driverless AI /data folder.

You can provide a training, validation, and testing dataset for an experiment. The validation and testing dataset are optional. In this example, we will provide only training and testing.

In [3]: train_path = '/data/Kaggle/CreditCard/CreditCard-train.csv'
   test_path = '/data/Kaggle/CreditCard/CreditCard-test.csv'

   train = h2oai.create_dataset_sync(train_path)
   test = h2oai.create_dataset_sync(test_path)
Equivalent Steps in Driverless: Uploading Train & Test CSV Files

![H2O.ai](https://h2o.ai)

Fig. 4: Equivalent Steps in Driverless: Upload Train & Test CSV Files

3. Set Experiment Parameters

We will now set the parameters of our experiment. Some of the parameters include:

- **Target Column**: The column we are trying to predict.
- **Dropped Columns**: The columns we do not want to use as predictors such as ID columns, columns with data leakage, etc.
- **Weight Column**: The column that indicates the per row observation weights. If None, each row will have an observation weight of 1.
- **Fold Column**: The column that indicates the fold. If None, the folds will be determined by Driverless AI.
- **Is Time Series**: Whether or not the experiment is a time-series use case.

For information on the experiment settings, refer to the *Experiment Settings*.

For this example, we will be predicting "**default payment next month**". The parameters that control the experiment process are: accuracy, time, and interpretability. We can use the `get_experiment_preview_sync` function to get a sense of what will happen during the experiment.

We will start out by seeing what the experiment will look like with accuracy, time, and interpretability all set to 5.

```python
In [4]: target="default payment next month"
exp_preview = h2oai.get_experiment_preview_sync(dataset_key= train.key, validset_key='', classification=True,
dropped_cols = [], target_col=target, is_time
enable_gpus = True,
accuracy = 5, time = 5, interpretability = 5,
config_overrides = None)
```

```
exp_preview
```

```
Out[4]: ['ACCURACY [5/10]:'
'- Training data size: *23,999 rows, 25 cols*',
'- Feature evolution: *[LightGBM, XGBoost]*, *1/3 validation split*',
'- Final pipeline: *Ensemble (4 models), 4-fold CV*',
''
'TIME [5/10]:'
'- Feature evolution: *8 individuals*, up to *54 iterations*',
'- Early stopping: After *10* iterations of no improvement',
''
'INTERPRETABILITY [5/10]:'
'- Feature pre-pruning strategy: None',
```
Using Driverless AI, Release 1.6.2

'- XGBoost Monotonicity constraints: disabled',
'- Feature engineering search space (where applicable): [Clustering, Date, FrequencyEncoding, ... OneHotEncoding, TargetEncoding, Text, TextBiGRU, TextCNN, TextCharCNN, TextLin, TruncatedSVD, WeightOfEvidence]

'[LightGBM, XGBoost] models to train:',
'- Model and feature tuning: *16*',
'- Feature evolution: *208*',
'- Final pipeline: *4*',
'
'Estimated runtime: *minutes*]

With these settings, the Driverless AI experiment will train about 228 models: * 16 for model and feature tuning * 208 for feature evolution * 4 for the final pipeline

When we start the experiment, we can either:

- specify parameters
- use Driverless AI to suggest parameters

Driverless AI can suggest the parameters based on the dataset and target column. Below we will use the "get_experiment_tuning_suggestion" to see what settings Driverless AI suggests.

In [5]: # let Driverless suggest parameters for experiment
   params = h2oai.get_experiment_tuning_suggestion(dataset_key = train.key, target_col = target, is_classification = True, is_time_series = False, config_overrides = None)

In [6]: params.dump()

Out[6]: {'dataset_key': 'pepipuci',
   'resumed_model_key': '',
   'target_col': 'default payment next month',
   'weight_col': '',
   'fold_col': '',
   'orig_time_col': '',
   'time_col': '',
   'is_classification': True,
   'cols_to_drop': [],
   'validset_key': '',
   'testset_key': '',
   'enable_gpus': True,
   'seed': False,
   'accuracy': 6,
   'time': 4,
   'interpretability': 6,
   'scorer': 'AUC',
   'time_groups_columns': [],
   'time_period_in_seconds': None,
   'num_prediction_periods': None,
   'num_gap_periods': None,
   'is_timeseries': False,
   'configOverrides': None}

Driverless AI has found that the best parameters are to set "accuracy = 6", "time = 4", "interpretability = 6". It has selected "AUC" as the scorer (this is the default scorer for binomial problems).
Equivalent Steps in Driverless: Set the Knobs, Configuration & Launch

![H2O.ai Experiment](image)

What do these settings mean?

- **ACCURACY**
  - Training data size: 22,999 rows, 23 cols
  - Feature evolution: [LightGBM, XGBoost], 1/3 validation split
  - Final pipeline: Ensemble (10 models), 5-fold CV

- **TIME**
  - Feature evolution: 8 individuals, up to 48 iterations
  - Early stopping: After 8 iterations of no improvement

- **INTERPRETABILITY**
  - Feature inference: strategy: FS
  - XGBoost: handle missing values: enabled

- **EXPLORE**
  - Feature selection: 10
  - Final pipeline: 10

- **ESTIMATED RUNTIME**
  - [Min] minutes

**Fig. 5: Equivalent Steps in Driverless: Set the Knobs**

### 4. Launch Experiment: Feature Engineering + Final Model Training

Launch the experiment using the parameters that Driverless AI suggested along with the testset, scorer, and seed that were added. We can launch the experiment with the suggested parameters or create our own.

In [7]:
```python
experiment = h2oai.start_experiment_sync(dataset_key=train.key,
                                        testset_key=test.key,
                                        target_col=target,
                                        is_classification=True,
                                        accuracy=6,
                                        time=4,
                                        interpretability=6,
                                        scorer="AUC",
                                        enable_gpus=True,
                                        seed=1234,
                                        cols_to_drop=['ID'])
```
5. Examine Experiment

View the final model score for the validation and test datasets. When feature engineering is complete, an ensemble model can be built depending on the accuracy setting. The experiment object also contains the score on the validation and test data for this ensemble model. In this case, the validation score is the score on the training cross-validation predictions.

In [8]:
```
print("Final Model Score on Validation Data: " + str(round(experiment.valid_score, 3)))
print("Final Model Score on Test Data: " + str(round(experiment.test_score, 3)))
```

Final Model Score on Validation Data: 0.779
Final Model Score on Test Data: 0.797

The experiment object also contains the scores calculated for each iteration on bootstrapped samples on the validation data. In the iteration graph in the UI, we can see the mean performance for the best model (yellow dot) and +/- 1 standard deviation of the best model performance (yellow bar).

This information is saved in the experiment object.
Using Driverless AI, Release 1.6.2

In [11]: import matplotlib.pyplot as plt

# Add scores from experiment iterations
iteration_data = h2oaai.list_model_iteration_data(experiment.key, 0, len(experiment.iteration_data))
iterations = list(map(lambda iteration: iteration.iteration, iteration_data))
scores_mean = list(map(lambda iteration: iteration.score_mean, iteration_data))
scores_sd = list(map(lambda iteration: iteration.score_sd, iteration_data))

# Add score from final ensemble
iterations = iterations + [max(iterations) + 1]
scores_mean = scores_mean + [experiment.valid_score]
scores_sd = scores_sd + [experiment.valid_score_sd]

plt.figure()
plt.errorbar(iterations, scores_mean, yerr=scores_sd, color = "y",
    ecolor='yellow', fmt = '--o', elinewidth = 4, alpha = 0.5)
plt.xlabel("Iteration")
plt.ylabel("AUC")
plt.ylim([0.65, 0.82])
plt.show();
6. Download Results

Once an experiment is complete, we can see that the UI presents us options of downloading the:

- predictions
  - on the (holdout) train data
  - on the test data
- experiment summary - summary of the experiment including feature importance

We will show an example of downloading the test predictions below. Note that equivalent commands can also be run for downloading the train (holdout) predictions.

```python
In [12]: h2oai.download(src_path=experiment.test_predictions_path, dest_dir=".")
Out[12]: './test_preds.csv'
In [13]: test_preds = pd.read_csv('./test_preds.csv')
test_preds.head()
Out[13]:
default payment next month.0  default payment next month.1
0   0.365548                  0.634452
1   0.872698                  0.127302
2   0.945317                  0.054683
3   0.504811                  0.495189
4   0.886989                  0.113011
```
34.2.3 Build an Experiment in Web UI and Access Through Python

It is also possible to use the Python API to examine an experiment that was started through the Web UI using the experiment key.

1. Get pointer to experiment

You can get a pointer to the experiment by referencing the experiment key in the Web UI.

In [28]: # Get list of experiments
   
   experiment_list = list(map(lambda x: x.key, h2oai.list_models(offset=0, limit=100)))
   
   experiment_list

Out[28]: ['hesenifo', 'hefihuci']

In [14]: # Get pointer to experiment
   
   experiment = h2oai.get_model_job(experiment_list[0]).entity

34.2.4 Score on New Data

You can use the Python API to score on new data. This is equivalent to the SCORE ON ANOTHER DATASET button in the Web UI. The example below scores on the test data and then downloads the predictions.

Pass in any dataset that has the same columns as the original training set. If you passed a test set during the H2OAI model building step, the predictions already exist. Its path can be found with "experiment.test_predictions_path".

1. Score Using the H2OAI Model

The following shows the predicted probability of default for each record in the test.

In [15]: prediction = h2oai.make_prediction_sync(experiment.key, test_path, output_margin = False, pred_contribs = False)
   
   pred_path = h2oai.download(prediction.predictions_csv_path, '.')
   
   pred_table = pd.read_csv(pred_path)
   
   pred_table.head()

Out[15]:

<table>
<thead>
<tr>
<th>default payment next month.0</th>
<th>default payment next month.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.365548</td>
<td>0.634452</td>
</tr>
<tr>
<td>0.872698</td>
<td>0.127302</td>
</tr>
<tr>
<td>0.945317</td>
<td>0.054683</td>
</tr>
<tr>
<td>0.504811</td>
<td>0.495189</td>
</tr>
<tr>
<td>0.886989</td>
<td>0.113011</td>
</tr>
</tbody>
</table>

We can also get the contribution each feature had to the final prediction by setting pred_contribs = True. This will give us an idea of how each feature effects the predictions.
```python
In [16]: prediction_contributions = h2oai.make_prediction_sync(experiment.key, test_path,
                 output_margin = False, pred_contribs = True)
pred_contributions_path = h2oai.download(prediction_contributions.predictions_csv_path, '.
pred_contributions_table = pd.read_csv(pred_contributions_path)
pred_contributions_table.head()
```

```
Out[16]:
contrib_0_AGE  contrib_1_BILL_AMT1
0  0.000470  -0.007112
1  -0.000368  -0.024320
2  -0.000435  -0.045927
3  -0.000821  -0.009697
4  -0.000567  -0.031249

contrib_100_NumToCatWoE:BILL_AMT1:BILL_AMT3:PAY_6:PAY_AMT2.0
0  -0.002012
1  -0.004683
2  -0.005945
3  -0.003584
4   0.005642

contrib_102_NumToCatWoE:BILL_AMT2.0
0  0.001037
1  0.001353
2  -0.000975
3  -0.002837
4  0.001110

contrib_103_NumToCatTE:BILL_AMT1:PAY_0:PAY_2:PAY_3:PAY_AMT1.0
0   0.002586
1  -0.007675
2  -0.010465
3  -0.002245
4  -0.021471

contrib_104_InteractionMul:PAY_0:PAY_4  contrib_105_NumToCatWoE:PAY_2.0
0  -0.000042  -0.002168
1   0.000027   0.00454
2  -0.000121  0.000637
3   0.002572  -0.003896
4  -0.000098  -0.000162

contrib_106_ClusterTE:ClusterID7:PAY_0:PAY_2.0
0  0.000025
1  -0.000113
2  -0.000349
3   0.000225
4  -0.000386

contrib_108_NumToCatTE:PAY_0:PAY_2:PAY_3:PAY_5.0
0  0.001883
1  -0.000723
2  -0.002303
3   0.001469
4   0.002160

contrib_113_InteractionSub:PAY_5:PAY_AMT3
0  -0.007623
1  -0.023062
2   0.020167
```

390 Chapter 34. Appendix A: The Python Client
3 0.005805 ...
4 -0.007287 ...

ccontrib_93_interactionSub:PAY_2:PAY_AMT2 \ 0 -0.000425 1 -0.002609 2 -0.003587 3 0.004117 4 0.009114

ccontrib_93_interactionSub:PAY_5:PAY_AMT1 \ 0 0.004040 1 0.003854 2 0.000937 3 0.005500 4 0.007328

ccontrib_93_interactionSub:PAY_5:PAY_AMT2 \ 0 0.000640 1 -0.001576 2 -0.007253 3 0.005881 4 0.008749

ccontrib_93_interactionSub:PAY_AMT1:PAY_AMT2 \ 0 0.003899 1 0.002270 2 -0.006040 3 0.002649 4 0.004359

ccontrib_95_CVCatNumEnc:LIMIT_BAL:PAY_0.sd \ 0 0.002187 1 0.001527 2 0.000108 3 0.001159 4 0.002195

ccontrib_95_CVCatNumEnc:LIMIT_BAL:PAY_3.sd \ 0 -0.002022 1 0.001040 2 -0.000666 3 0.002523 4 -0.001004

ccontrib_97_NumCatTE:PAY_0:PAY_2:PAY_5.0 \ 0 0.231381 1 -0.059638 2 -0.067213 3 0.199952 4 -0.054983

ccontrib_99_CVCatNumEnc:PAY_6:PAY_2.mean \ 0 -0.016387 1 -0.005219 2 -0.005032 3 0.041568 4 -0.006999
Using Driverless AI, Release 1.6.2

```python
contrib_99_CVCatNumEnc:PAY_6:PAY_AMT2.mean contrib_bias
0  -0.009169 -1.007552
1  -0.004099 -1.007552
2  -0.008155 -1.007552
3   0.022907 -1.007552
4   0.004216 -1.007552
```

[5 rows x 205 columns]

We will examine the contributions for our first record more closely.

```python
In [17]: contrib = pd.DataFrame(pred_contributions_table.iloc[0][1:], columns = ["contribution"]
contrib["abs_contribution"] = contrib.contribution.abs()
contrib.sort_values(by="abs_contribution", ascending=False)[["contribution"]).head()
```

```python
Out[17]:
contrib["abs_contribution"]
contrib_bias       -1.007552
contrib_34_ClusterTE:ClusterID61:PAY_0:PAY_3:PAY_AMT1:PAY_4:PAY_5.0  0.320730
contrib_97_NumCatTE:PAY_0:PAY_2:PAY_5.0                           0.231381
contrib_92_WoE:PAY_0:PAY_2:PAY_5.0                               0.134660
contrib_40_WoE:PAY_0:PAY_2:PAY_3.0                              0.124549
```

The clusters from this customer’s: PAY_0, PAY_3, PAY_5, and PAY_AMT1 had the greatest impact on their prediction. Since the contribution is positive, we know that it increases the probability that they will default.

### 34.2.5 Model Diagnostics on New Data

You can use the Python API to also perform model diagnostics on new data. This is equivalent to the Model Diagnostics tab in the Web UI.

![Model Diagnostics Setup](image)

**Fig. 10: Model Diagnostics Setup**

1. Run model diagnostics on new data with H2OAI model

The example below performs model diagnostics on the test dataset but any data with the same columns can be selected.

```python
In [18]: test_diagnostics = h2oai.make_model_diagnostic_sync(experiment.key, test.key)
In [19]: [x.scorer, x.score for x in test_diagnostics.scores]
```

```python
Out[19]:
[{'scorer': 'GINI', 'score': 0.5947506892761234},
 {'scorer': 'MCC', 'score': 0.44238638377750966},
 {'scorer': 'F05', 'score': 0.5951818611242324},
 {'scorer': 'F1', 'score': 0.5528394133132757},
]```
Here is the same model diagnostics displayed in the UI:

![Model Diagnostics Setup](image)

### 34.2.6 Run Model Interpretation

Once we have completed an experiment, we can interpret our H2OAI model. Model Interpretability is used to provide model transparency and explanations. More information on Model Interpretability can be found here: [http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/interpreting.html](http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/interpreting.html).

#### 1. Run Model Interpretation on the Raw Data

We can run the model interpretation in the Python client as shown below. By setting the parameter, `use_raw_features` to True, we are interpreting the model using only the raw features in the data. This will not use the engineered features we saw in our final model’s features to explain the data.

```python
In [20]: mli_experiment = h2oai.run_interpretation_sync(dai_model_key = experiment.key,
                                              dataset_key = train.key,
                                              target_col = target,
                                              use_raw_features = True)
```

This is equivalent to clicking **Interpret this Model on Original Features** in the UI once the experiment has completed.
Once our interpretation is finished, we can navigate to the MLI tab in the UI to see our interpreted model.

We can also see the list of interpretations using the Python Client:

```python
In [22]: # Get list of interpretations
mli_list = list(map(lambda x: x.key, h2oai.list_interpretations(offset=0, limit=100)))
mli_list
Out[22]: ['dukicana']
```

2. Run Model Interpretation on External Model Predictions

Model Interpretation does not need to be run on a Driverless AI experiment. We can also train an external model and run Model Interpretability on the predictions. In this next section, we will walk through the steps to interpret an external model.
Train External Model

We will begin by training a model with scikit-learn. Our end goal is to use Driverless AI to interpret the predictions made by our scikit-learn model.

In [24]: # Dataset must be located where Python client is running
   train_pd = pd.read_csv(train_path)

In [25]: from sklearn.ensemble import GradientBoostingClassifier
   predictors = list(set(train_pd.columns) - set([target]))
   gbm_model = GradientBoostingClassifier(random_state=10)
   gbm_model.fit(train_pd[predictors], train_pd[target])

Out[25]: GradientBoostingClassifier(criterion='friedman_mse', init=None,
   learning_rate=0.1, loss='deviance', max_depth=3,
   max_features=None, max_leaf_nodes=None,
   min_impurity_decrease=0.0, min_impurity_split=None,
   min_samples_leaf=1, min_samples_split=2,
   min_weight_fraction_leaf=0.0, n_estimators=100,
   presort='auto', random_state=10, subsample=1.0, verbose=0,
   warm_start=False)

In [26]: predictions = gbm_model.predict_proba(train_pd[predictors])
   predictions[0:5]

Out[26]: array([[0.38111179, 0.61888821],
   [0.44396186, 0.55603814],
   [0.91738328, 0.08261672],
   [0.88780536, 0.11219464],
   [0.80028008, 0.19971992]])

Interpret on External Predictions

Now that we have the predictions from our scikit-learn GBM model, we can call Driverless AI's "h2o_ai.run_interpretation_sync" to create the interpretation screen.

In [27]: train_gbm_path = "./CreditCard-train-gbm_pred.csv"
   predictions = pd.concat([train_pd, pd.DataFrame(predictions[:, 1], columns = ["p1")], axis = 1)
   predictions.to_csv(path_or_buf=train_gbm_path, index = False)
   train_gbm_pred = h2oai.upload_dataset_sync(train_gbm_path)
   mli_external = h2oai.run_interpretation_sync(dai_model_key = ",
   # no experiment key
dataset_key = train_gbm_pred.key,
   target_col = target,
prediction_col = "p1")

We can also run Model Interpretability on an external model in the UI as shown below:
In [53]: # Get list of interpretations
    mli_list = list(map(lambda x: x.key, h2oai.list_interpretations(offset=0, limit=100)))
    mli_list
Out[53]: ['fawinusa', 'dukicono']

34.2.7 Build Scoring Pipelines

In our last section, we will build the scoring pipelines from our experiment. There are two scoring pipeline options:

- Python Scoring Pipeline: requires Python runtime
- MOJO Scoring Pipeline: requires Java runtime

Documentation on the scoring pipelines is provided here: http://docs.h2o.ai/driverless-ai/latest-stable/docs/userguide/python-mojo-pipelines.html.
The experiment screen shows two scoring pipeline buttons: **Download Python Scoring Pipeline** or **Build MOJO Scoring Pipeline**. Driverless AI determines if any scoring pipeline should be automatically built based on the config.toml file. In this example, we have run Driverless AI with the settings:

```plaintext
# Whether to create the Python scoring pipeline at the end of each experiment
make_python_scoring_pipeline = true

# Whether to create the MOJO scoring pipeline at the end of each experiment
# Note: Not all transformers or main models are available for MOJO (e.g. no gblinear, main model)
make_mojo_scoring_pipeline = false
```

Therefore, only the Python Scoring Pipeline will be built by default.

### 1. Build Python Scoring Pipeline

The Python Scoring Pipeline has been built by default based on our config.toml settings. We can get the path to the Python Scoring Pipeline in our experiment object.

```
In [30]: experiment.scoring_pipeline_path
Out[30]: 'h2oai_experiment_mecifore/scoring_pipeline/scorer.zip'
```

We can also build the Python Scoring Pipeline - this is useful if the "make_python_scoring_pipeline" option was set to false.

```
In [31]: python_scoring_pipeline = h2oai.build_scoring_pipeline_sync(experiment.key)
In [32]: python_scoring_pipeline.file_path
Out[32]: 'h2oai_experiment_mecifore/scoring_pipeline/scorer.zip'
```

Now we will download the scoring pipeline zip file.
In [33]: h2oai.download(python_scoring_pipeline.file_path, dest_dir=".")
Out[33]: './scorer.zip'

2. Build MOJO Scoring Pipeline

The MOJO Scoring Pipeline has not been built by default because of our config.toml settings. We can build the MOJO Scoring Pipeline using the Python client. This is equivalent to selecting the Build MOJO Scoring Pipeline on the experiment screen.

In [34]: mojo_scoring_pipeline = h2oai.build_mojo_pipeline_sync(experiment.key)
In [35]: mojo_scoring_pipeline.file_path
Out[35]: 'h2oai_experiment_mecifore/mojo_pipeline/mojo.zip'

Now we can download the scoring pipeline zip file.

In [36]: h2oai.download(mojo_scoring_pipeline.file_path, dest_dir=".")
Out[36]: './mojo.zip'

Once the MOJO Scoring Pipeline is built, the Build MOJO Scoring Pipeline changes to Download MOJO Scoring Pipeline.

Fig. 16: Equivalent Steps in Driverless: Download MOJO
34.3 Driverless AI - Training Time Series Model

The purpose of this notebook is to show an example of using Driverless AI to train a time series model. Our goal will be to forecast the Weekly Sales for a particular Store and Department for the next week. The data used in this notebook is from the: Walmart Kaggle Competition.

34.3.1 Workflow

1. Import data into Python
2. Format data for Time Series
3. Upload data to Driverless AI
4. Launch Driverless AI Experiment
5. Evaluate model performance

34.3.2 Step 1: Import Data

We will begin by importing our data using pandas. We are going to first work with the data in Python to correctly format it for a Driverless AI time series use case.

In [4]: import pandas as pd

sales_data = pd.read_csv("./walmart_train.csv")
sales_data.head()

Out[4]:
<table>
<thead>
<tr>
<th>Store</th>
<th>Dept</th>
<th>Date</th>
<th>Weekly_Sales</th>
<th>Temperature</th>
<th>Fuel_Price</th>
<th>MarkDown1</th>
<th>MarkDown2</th>
<th>MarkDown3</th>
<th>MarkDown4</th>
<th>MarkDown5</th>
<th>CPI</th>
<th>Unemployment</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2010-02-05</td>
<td>24924.50</td>
<td>42.31</td>
<td>2.572</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>211.096358</td>
<td>8.106</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>2010-02-05</td>
<td>50605.27</td>
<td>42.31</td>
<td>2.572</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>211.096358</td>
<td>8.106</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>2010-02-05</td>
<td>13740.12</td>
<td>42.31</td>
<td>2.572</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>211.096358</td>
<td>8.106</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2010-02-05</td>
<td>39954.04</td>
<td>42.31</td>
<td>2.572</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>211.096358</td>
<td>8.106</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>2010-02-05</td>
<td>32229.38</td>
<td>42.31</td>
<td>2.572</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>211.096358</td>
<td>8.106</td>
<td></td>
</tr>
</tbody>
</table>

IsHoliday sample_weight
| False | 1
| False | 1
| False | 1
| False | 1
| False | 1

In [5]: # Convert Date column to datetime
sales_data["Date"] = pd.to_datetime(sales_data["Date"], format="%Y-%m-%d")
34.3.3 Step 2: Format Data for Time Series

The data has one record per Store, Department, and Week. Our goal for this use case will be to forecast the total sales for the next week.

The only features we should use as predictors are ones that we will have available at the time of scoring. Features like the Temperature, Fuel Price, and Unemployment will not be known in advance. Therefore, before we start our Driverless AI experiments, we will choose to use the previous week’s Temperature, Fuel Price, Unemployment, and CPI attributes. This information we will know at time of scoring.

```
In [6]: lag_variables = ['Temperature', 'Fuel_Price', 'CPI', 'Unemployment']
dai_data = sales_data.set_index(['Date', 'Store', 'Dept'])
lagged_data = dai_data.loc[:, lag_variables].groupby(level=['Store', 'Dept']).shift(1)
```

```
In [8]: # Join lagged predictor variables to training data
dai_data = dai_data.join(lagged_data.rename(columns=lambda x: x + '_lag'))
```

```
In [9]: # Drop original predictor variables - we do not want to use these in the model
dai_data = dai_data.drop(lagged_data, axis=1)
dai_data = dai_data.reset_index()
```

```
In [10]: dai_data.head()
```

```
Out[10]:

<table>
<thead>
<tr>
<th></th>
<th>Date</th>
<th>Store</th>
<th>Dept</th>
<th>Weekly_Sales</th>
<th>MarkDown1</th>
<th>MarkDown2</th>
<th>MarkDown3</th>
<th>MarkDown4</th>
<th>MarkDown5</th>
<th>IsHoliday</th>
<th>sample_weight</th>
<th>Temperature_lag</th>
<th>Fuel_Price_lag</th>
<th>CPI_lag</th>
<th>Unemployment_lag</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2010-02-05</td>
<td>1</td>
<td>1</td>
<td>24924.50</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>False</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>1</td>
<td>2010-02-05</td>
<td>1</td>
<td>2</td>
<td>50605.27</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>False</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>2</td>
<td>2010-02-05</td>
<td>1</td>
<td>3</td>
<td>13740.12</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>False</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>3</td>
<td>2010-02-05</td>
<td>1</td>
<td>4</td>
<td>39954.04</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>False</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
<tr>
<td>4</td>
<td>2010-02-05</td>
<td>1</td>
<td>5</td>
<td>32229.38</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>-1.0</td>
<td>False</td>
<td>1</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
<td>NaN</td>
</tr>
</tbody>
</table>
```

Now that our training data is correctly formatted, we can run a Driverless AI experiment to forecast the next week’s sales.

```
In [ ]: def dai_moving_window(dataset, train_len, test_len, target, predictors, day_number_col,
                        time_group_cols, accuracy, time, interpretability):

    windows = get_moving_windows(dataset, train_len, test_len, day_number_col)

    # Split the data by the window
    forecast_predictions = pd.DataFrame([])
    model_descriptions = []
    model_features = []

    for window in windows:
        train_data = dataset[day_number_col] >= window.get("train_start_index") &
                    dataset[day_number_col] <= window.get("train_end_index")

    # Run the Driverless AI experiment
    prediction = run_driverless_ai(experiment, train_data, target, predictors)
```

```
```

```
```
test_data = dataset[(dataset[day_number_col] >= window.get("test_start_index")) & (dataset[day_number_col] <= window.get("test_end_index"))]

# Get the Driverless AI forecast predictions
preds, desc, features = dai_get_forecast(train_data, test_data, predictors, target, day_number_col, time_group_cols, accuracy, time)
forecast_predictions = forecast_predictions.append(preds)
model_descriptions = model_descriptions + [desc]
model_features = model_features + [features]

return forecast_predictions, model_descriptions, model_features

34.3.4 Step 3: Upload Data to Driverless AI

We will split out data into two pieces: training and test (which consists of the last week of data).

In [11]: train_data = dai_data.loc[dai_data["Date"] < "2012-10-26"]
       test_data = dai_data.loc[dai_data["Date"] == "2012-10-26"]

To upload the datasets, we will sign into Driverless AI.

In [13]: from h2oai_client import Client, ModelParameters

In [ ]: address = 'http://ip_where_driverless_is_running:12345'
       username = 'username'
       password = 'password'
       h2oai = Client(address = address, username = username, password = password)

# make sure to use the same user name and password when signing in through the GUI

In [12]: train_path = "./train_data.csv"
       test_path = "./test_data.csv"

   train_data.to_csv(train_path, index = False)
   test_data.to_csv(test_path, index = False)

In [13]: # Add datasets to Driverless AI
   train_dai = h2oai.upload_dataset_sync(train_path)
   test_dai = h2oai.upload_dataset_sync(test_path)

Equivalent Steps in Driverless: Uploading Train & Test CSV Files

Fig. 17: Equivalent Steps in Driverless: Upload Train & Test CSV Files

34.3. Driverless AI - Training Time Series Model
34.3.5 Step 4: Launch Driverless AI Experiment

We will now launch the Driverless AI experiment. To do that we will need to specify the parameters for our experiment. Some of the parameters include:

- Target Column: The column we are trying to predict.
- Dropped Columns: The columns we do not want to use as predictors such as ID columns, columns with data leakage, etc.
- Is Time Series: Whether or not the experiment is a time-series use case.
- Time Column: The column that contains the date/date-time information.
- Time Group Columns: The categorical columns that indicate how to group the data so that there is one time series per group. In our example, our Time Groups Columns are Store and Dept. Each Store and Dept, corresponds to a single time series.
- Number of Prediction Periods: How far in the future do we want to predict?
- Number of Gap Periods: After how many periods can we start predicting? If we assume that we can start forecasting right after the training data ends, then the Number of Gap Periods will be 0.

For this experiment, we want to forecast next week's sales for each Store and Dept. Therefore, we will use the following time series parameters:

- Time Group Columns: [Store, Dept]
- Number of Prediction Periods: 1
- Number of Gap Periods: 0

```python
In [14]: experiment = h2oai.start_experiment_sync(dataset_key=train_dai.key,
                                                testset_key = test_dai.key,
                                                target_col="Weekly_Sales",
                                                is_classification=False,
                                                cols_to_drop = ["sample_weight"],
                                                accuracy=5,
                                                time=3,
                                                interpretability=1,
                                                scorer="RMSE",
                                                enable_gpus=True,
                                                seed=1234,
                                                time_col = "Date",
                                                time_groups_columns = ["Store", "Dept"],
                                                num_prediction_periods = 1,
                                                num_gap_periods = 0)
```
34.3.6 Step 5. Evaluate Model Performance

Now that our experiment is complete, we can view the model performance metrics within the experiment object.

```python
In [15]: print("Validation RMSE: ${:,.0f}".format(experiment.valid_score))
print("Test RMSE: ${:,.0f}".format(experiment.test_score))
```

Validation RMSE: $2,199
Test RMSE: $2,433

We can also plot the actual versus predicted values from the test data.

```python
In [16]: %matplotlib inline
import matplotlib.pyplot as plt
plt.scatter(experiment.test_act_vs_pred.predicted, experiment.test_act_vs_pred.actual)
plt.plot([0, max(experiment.test_act_vs_pred.predicted)],[0, max(experiment.test_act_vs_pred.actual)], 'b--',)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```
Lastly, we can download the test predictions from Driverless AI and examine the forecasted sales vs actual for a selected store and department.

```python
In [17]: preds_path = h2oai.download(src_path=experiment.test_predictions_path, dest_dir=".")
forecast_predictions = pd.read_csv(preds_path)
forecast_predictions.columns = ["predicted_Weekly_Sales"]

actual = test_data[["Date", "Store", "Dept", "Weekly_Sales"]].reset_index(drop = True)
forecast_predictions = pd.concat([actual, forecast_predictions], axis = 1)
forecast_predictions.head()
```

```
Out[17]:

<table>
<thead>
<tr>
<th>Date</th>
<th>Store</th>
<th>Dept</th>
<th>Weekly_Sales</th>
<th>predicted_Weekly_Sales</th>
</tr>
</thead>
<tbody>
<tr>
<td>2012-10-26</td>
<td>1</td>
<td>1</td>
<td>27390.81</td>
<td>29016.900</td>
</tr>
<tr>
<td>2012-10-26</td>
<td>1</td>
<td>2</td>
<td>43134.88</td>
<td>42700.937</td>
</tr>
<tr>
<td>2012-10-26</td>
<td>1</td>
<td>3</td>
<td>9350.90</td>
<td>9351.535</td>
</tr>
<tr>
<td>2012-10-26</td>
<td>1</td>
<td>4</td>
<td>36292.60</td>
<td>34871.164</td>
</tr>
<tr>
<td>2012-10-26</td>
<td>1</td>
<td>5</td>
<td>25846.94</td>
<td>23360.783</td>
</tr>
</tbody>
</table>
```

```python
In [26]: selected_ts = sales_data.loc[(sales_data["Store"] == 1) & (sales_data["Dept"] == 1)].tail(n = 51)
selected_ts_forecast = forecast_predictions.loc[(forecast_predictions["Store"] == 1) &
    (forecast_predictions["Dept"] == 1)]

In [27]: # Plot the forecast of a select store and department

    import matplotlib.dates as mdates

    years = mdates.MonthLocator()
    yearsFmt = mdates.DateFormatter('%b')
    fig, ax = plt.subplots()
    ax.plot(selected_ts["Date"], selected_ts["Weekly_Sales"], label = "Actual")
    ax.plot(selected_ts_forecast["Date"], selected_ts_forecast["predicted_Weekly_Sales"], marker='o', label = "Predicted")
    ax.xaxis.set_major_locator(years)
    ax.xaxis.set_major_formatter(yearsFmt)
```

```
Out[27]:

```
```
plt.legend(loc='upper left')
plt.show()

In [ ]:

### 34.4 Driverless AI - Time Series Recipes with Rolling Window

The purpose of this notebook is to show an example of using Driverless AI to train experiments on different subsets of data. This would result in a collection of forecasted values that can be evaluated. The data used in this notebook is a public dataset: S+P 500 Stock Data. In this example, we are using the `all_stocks_5yr.csv` dataset.

#### 34.4.1 Workflow

1. Import data into Python
2. Create function that slices data by index
3. For each slice of data:
   • import data into Driverless AI
   • train an experiment
   • combine test predictions

#### 34.4.2 Import Data

We will begin by importing our data using pandas.

In [1]: import pandas as pd

    stock_data = pd.read_csv("./all_stocks_5yr.csv")
    stock_data.head()
Using Driverless AI, Release 1.6.2

```
Out[1]:
<table>
<thead>
<tr>
<th>date</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>volume</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-02-08</td>
<td>15.07</td>
<td>15.12</td>
<td>14.63</td>
<td>14.75</td>
<td>8407500</td>
<td>AAL</td>
</tr>
<tr>
<td>2013-02-11</td>
<td>14.89</td>
<td>15.01</td>
<td>14.26</td>
<td>14.46</td>
<td>8882000</td>
<td>AAL</td>
</tr>
<tr>
<td>2013-02-12</td>
<td>14.45</td>
<td>14.51</td>
<td>14.10</td>
<td>14.27</td>
<td>8126000</td>
<td>AAL</td>
</tr>
</tbody>
</table>
```

```
In [2]:
# Convert Date column to datetime
stock_data["date"] = pd.to_datetime(stock_data["date"], format="%Y-%m-%d")

We will add a new column which is the index. We will use this later on to do a rolling window of training and testing.
We will use this index instead of the actual date because this data only occurs on weekdays (when the stock market is opened). When you use Driverless AI to perform a forecast, it will forecast the next \textit{n} days. In this particular case, we never want to forecast Saturday’s and Sunday’s. We will instead treat our time column as the index of the record.

```
In [3]:
# Split the data by time
dates_index = pd.DataFrame(sorted(stock_data["date"].unique()), columns = ["date"])
dates_index["index"] = range(len(dates_index))
stock_data = pd.merge(stock_data, dates_index, on = "date")
stock_data.head()
```

```
Out[3]:
<table>
<thead>
<tr>
<th>date</th>
<th>open</th>
<th>high</th>
<th>low</th>
<th>close</th>
<th>volume</th>
<th>Name</th>
<th>index</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013-02-08</td>
<td>15.0700</td>
<td>15.1200</td>
<td>14.6300</td>
<td>14.7500</td>
<td>8407500</td>
<td>AAL</td>
<td>0</td>
</tr>
<tr>
<td>2013-02-08</td>
<td>67.7142</td>
<td>68.4014</td>
<td>66.8928</td>
<td>67.8542</td>
<td>158168416</td>
<td>AAPL</td>
<td>0</td>
</tr>
<tr>
<td>2013-02-08</td>
<td>78.3400</td>
<td>79.7200</td>
<td>78.0100</td>
<td>78.9000</td>
<td>1298137</td>
<td>AAP</td>
<td>0</td>
</tr>
<tr>
<td>2013-02-08</td>
<td>36.3700</td>
<td>36.4200</td>
<td>35.8250</td>
<td>36.2500</td>
<td>13858795</td>
<td>ABBV</td>
<td>0</td>
</tr>
<tr>
<td>2013-02-08</td>
<td>46.5200</td>
<td>46.8950</td>
<td>46.4600</td>
<td>46.8900</td>
<td>1232802</td>
<td>ABC</td>
<td>0</td>
</tr>
</tbody>
</table>
```

34.4.3 Create Moving Window Function

Now we will create a function that can split our data by time to create multiple experiments.

We will start by first logging into Driverless AI.

```
In [4]:
import h2oai_client
import numpy as np
import pandas as pd
# import h2o
import requests
import math
from h2oai_client import Client, ModelParameters

In [10]:
address = 'http://ip_where_driverless_is_running:12345'
username = 'username'
password = 'password'
h2oai = Client(address = address, username = username, password = password)
# make sure to use the same user name and password when signing in through the GUI
```

Our function will split the data into training and testing based on the training length and testing length specified by the user. It will then run an experiment in Driverless AI and download the test predictions.

```
In [11]:
def dai_moving_window(dataset, train_len, test_len, target, predictors, index_col, time_group_cols,
accuracy, time, interpretability):

    # Calculate windows for the training and testing data based on the train_len and test_len arguments
    num_dates = max(dataset[index_col])
    num_windows = (num_dates - train_len) // test_len

    windows = []
    for i in range(num_windows):
        train_start_id = i*test_len
```

406 Chapter 34. Appendix A: The Python Client
train_end_id = train_start_id + (train_len - 1)

test_start_id = train_end_id + 1

test_end_id = test_start_id + (test_len - 1)

window = {'train_start_index': train_start_id,
          'train_end_index': train_end_id,
          'test_start_index': test_start_id,
          'test_end_index': test_end_id}

windows.append(window)

# Split the data by the window
forecast_predictions = pd.DataFrame()
for window in windows:
    train_data = dataset[(dataset[index_col] >= window.get("train_start_index")) & (dataset[index_col] <= window.get("train_end_index"))]
    test_data = dataset[(dataset[index_col] >= window.get("test_start_index")) & (dataset[index_col] <= window.get("test_end_index"))]

    # Get the Driverless AI forecast predictions
    window_preds = dai_get_forecast(train_data, test_data, predictors, target, index_col, time, interpretability)
    forecast_predictions = forecast_predictions.append(window_preds)

return forecast_predictions

In [12]: def dai_get_forecast(train_data, test_data, predictors, target, index_col, time_group_cols, accuracy, time, interpretability):

    # Save dataset
    train_path = "./train_data.csv"
    test_path = "./test_data.csv"
    keep_cols = predictors + [target, index_col] + time_group_cols
    train_data[keep_cols].to_csv(train_path)
    test_data[keep_cols].to_csv(test_path)

    # Add datasets to Driverless AI
    train_dai = h2oai.upload_dataset_sync(train_path)
    test_dai = h2oai.upload_dataset_sync(test_path)

    # Run Driverless AI Experiment
    experiment = h2oai.start_experiment_sync(dataset_key = train_dai.key,
                                              testset_key = test_dai.key,
                                              target_col = target,
                                              cols_to_drop = [],
                                              is_classification = False,
                                              accuracy = accuracy,
                                              time = time,
                                              interpretability = interpretability,
                                              scorer = "RMSE",
                                              seed = 1234,
                                              time_col = index_col,
                                              time_groups_columns = time_group_cols,
                                              num_prediction_periods = test_data[index_col].nunique(),
                                              num_gap_periods = 0)

    # Download the predictions on the test dataset
    test_predictions_path = h2oai.download(experiment.test_predictions_path, "./"
test_predictions = pd.read_csv(test_predictions_path)
test_predictions.columns = ["Prediction"]

# Add predictions to original test data
keep_cols = [target, index_col] + time_group_cols
test_predictions = pd.concat([test_data[keep_cols].reset_index(drop=True), test_predictions], axis = 1)

return test_predictions

In [13]: predictors = ["Name", "index"]
target = "close"
index_col = "index"
time_group_cols = ["Name"]

In [ ]: # We will filter the dataset to the first 1030 dates for demo purposes
filtered_stock_data = stock_data[stock_data["index"] <= 1029]
forecast_predictions = dai_moving_window(filtered_stock_data, 1000, 3, target, predictors, index_col, time_group_cols,
accuracy = 1, time = 1, interpretability = 1)

In [25]: forecast_predictions.head()
Out[25]:
<table>
<thead>
<tr>
<th></th>
<th>close</th>
<th>index</th>
<th>Name</th>
<th>Prediction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>44.90</td>
<td>1000</td>
<td>AAL</td>
<td>48.050527</td>
</tr>
<tr>
<td>1</td>
<td>121.63</td>
<td>1000</td>
<td>AAPL</td>
<td>119.485352</td>
</tr>
<tr>
<td>2</td>
<td>164.63</td>
<td>1000</td>
<td>AAP</td>
<td>167.960700</td>
</tr>
<tr>
<td>3</td>
<td>60.43</td>
<td>1000</td>
<td>ABBV</td>
<td>60.784213</td>
</tr>
<tr>
<td>4</td>
<td>83.62</td>
<td>1000</td>
<td>ABC</td>
<td>86.939174</td>
</tr>
</tbody>
</table>

In [26]: # Calculate some error metric
mae = (forecast_predictions[target] - forecast_predictions["Prediction"]).abs().mean()
print("Mean Absolute Error: \$\{:.2f\}".format(mae))

Mean Absolute Error: $6.79

34.5 Driverless AI NLP Demo - Airline Sentiment Dataset

In this notebook, we will see how to use Driverless AI python client to build text classification models using the Airline sentiment twitter dataset.

Import the necessary python modules to get started including the Driverless AI client. If not already installed, please download the python client from Driverless AI GUI and install the same.

In [1]: import h2oai_client
import numpy as np
import pandas as pd
from sklearn import model_selection
from h2oai_client import Client

The below code downloads the twitter airline sentiment dataset and save it in the current folder.

In [2]: ! wget https://www.figure-eight.com/wp-content/uploads/2016/03/Airline-Sentiment-2-w-AA.csv
Resolving www.figure-eight.com (www.figure-eight.com)... 52.0.208.137, 52.3.39.167
Connecting to www.figure-eight.com (www.figure-eight.com)|52.0.208.137|:443... connected. HTTP request sent, awaiting response... 200 OK
Length: 3704908 (3.5M) [application/octet-stream]
Saving to: 'Airline-Sentiment-2-w-AA.csv'
Airline-Sentiment-2 100%[================================================================================] 3.53M 660KB/s in 5.5s
We can now split the dataset into train and test files so as to build models.

```python
train_al, test_al = model_selection.train_test_split(al, test_size=0.2, random_state=2018)
train_al.to_csv("train_airline_sentiment.csv", index=False)
```

The first step is to establish a connection to Driverless AI using `Client`. Please key in your credentials and the url address.

```python
h2o = Client(address='http://localhost:12345', username='h2oai', password='h2oai')
```

Read the train and test files into Driverless AI using the `create_dataset_sync` command.

```python
train = h2o.create_dataset_sync(train_path)
test = h2o.create_dataset_sync(test_path)
```

Now let us look at some basic information about the dataset. To check the number of columns and rows in the dataset.

```python
print('Train Dataset: ', len(train.columns), 'x', train.row_count)
print('Test Dataset: ', len(test.columns), 'x', test.row_count)
```

Train Dataset: 20 x 11712
Test Dataset: 20 x 2928

To get the names of the columns in the training set.

```python
[c.name for c in train.columns]
```

Out[7]: ['_unit_id', '_golden', '_unit_state', '_trusted_judgments', '_last_judgment_at', 'airline_sentiment', 'airline_sentiment:confidence', 'negativereason', 'negativereason:confidence', 'airline', 'airline_sentiment_gold', 'name', 'negativereason_gold', 'retweet_count', 'text', 'tweet_coord', 'tweet_created', 'tweet_id', 'tweet_location', 'user_timezone']

We just need two columns for our experiment. `text` which contains the text of the tweet and `airline_sentiment` which contains the sentiment of the tweet (target column). We can drop the remaining columns for this experiment. Let us get a preview for the same.

Also please set enable the tensorflow models by setting `enable_tensorflow="on"` if you have a GPU. This will help in creating the CNN based text features.
In [8]: exp_preview = h2o.get_experiment_preview_sync(
    dataset_key=train.key,
    validset_key='',
    target_col='airline_sentiment',
    classification=True,
    dropped_cols=["_unit_id", "_golden", "_unit_state", "_trusted_judgments", "_last_judgment_at",
                  "airline_sentiment:confidence", "negativereason", "negativereason:confidence",
                  "airline_sentiment_gold", "name", "negativereason_gold", "retweet_count",
                  "tweet_coord", "tweet_created", "tweet_id", "tweet_location", "user_timezone"],
    accuracy=6,
    time=4,
    interpretability=8,
    time_col='',
    enable_gpus=True,
    config_overrides='enable_tensorflow="on"'
)

exp_preview

Out[8]: ['ACCURACY [6/10]:',
       '- Training data size: *11,712 rows, 2 cols*',
       '- Feature evolution: *XGBoost*, *time-based validation*','
       '- Final pipeline: *XGBoost*','
       '',
       'TIME [4/10]:',
       '- Feature evolution: *4 individuals*, up to *52 iterations*',
       '- Early stopping: After *5* iterations of no improvement',
       '',
       'INTERPRETABILITY [8/10]:',
       '- Feature pre-pruning strategy: FS',
       '- Monotonicity constraints: enabled',
       '- Feature engineering search space (where applicable): [Date, Identity, Interactions, Lags, Text, TextCNN, WeightOfEvidence]',
       '',
       'XGBoost models to train:',
       '- Model and feature tuning: *72*',
       '- Feature evolution: *252*',
       '- Final pipeline: *1*',
       '',
       'Estimated max. total memory usage:',
       '- Feature engineering: *144.0MB*',
       '- GPU XGBoost: *8.0MB*']

Please note that the Text and TextCNN features are enabled for this experiment.

Now we can start the experiment.

In [53]: model = h2o.start_experiment_sync(
    dataset_key=train.key,
    testset_key=test.key,
    target_col='airline_sentiment',
    scorer=None,
    is_classification=True,
    cols_to_drop=["_unit_id", "_golden", "_unit_state", "_trusted_judgments", "_last_judgment_at",
                  "airline_sentiment:confidence", "negativereason", "negativereason:confidence",
                  "airline_sentiment_gold", "name", "negativereason_gold", "retweet_count",
                  "tweet_coord", "tweet_created", "tweet_id", "tweet_location", "user_timezone"],
    accuracy=6,
    time=2,
    interpretability=8,
    time_col='',
    enable_gpus=True,
In [89]: print('Modeling completed for model ' + model.key)
Modeling completed for model pakimeto

In [56]: print('Logs available at', model.log_file_path)
Logs available at h2oai_experiment_pakimeto/h2oai_experiment_logs_pakimeto.zip

We can download the predictions to the current folder.

In [58]: test_preds = h2o.download(model.test_predictions_path, '.')
   print('Test set predictions available at', test_preds)
Test set predictions available at ./test_preds.csv


