Using H2O Driverless AI

Patrick Hall, Megan Kurka & Angela Bartz

http://docs.h2o.ai

March 2018: Version 1.0.24

Using H2O Driverless AI
by Patrick Hall, Megan Kurka & Angela Bartz

Published by H2O.ai, Inc.
2307 Leghorn St.
Mountain View, CA 94043

March 2018: Version 1.0.24

Contents

Overview

H2O Driverless AI is an artificial intelligence (AI) platform that 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.

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 the NVIDIA DGX-1 for order-of-magnitude faster training.

This document describes how to use H2O Driverless AI and is updated periodically. To view the latest Driverless AI User Guide, please go to `http://docs.h2o.ai`.

For more information about Driverless AI, please see `https://www.h2o.ai/driverless-ai/`. For a third-party review, please see `https://www.infoworld.com/article/3236048/machine-learning/review-h2oai-automates-machine-learning.html`.

Citation

To cite this booklet, use the following: Hall, P., Kurka, M., and Bartz, A. (Jan 2018). *Using H2O Driverless AI*. `http://docs.h2o.ai`

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`.

Installing and Upgrading Driverless AI

Installation and upgrade steps are provided in the Driverless AI User Guide. 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 the older K80 and M60 GPUs available in EC2 are supported and very convenient, but not as fast.)

Driverless AI requires 10 GB of free disk space to run and will stop running if less than 10 GB is available. You should have lots of system CPU memory (64 GB or more) and free disk space (at least 30 GB and/or 10x your dataset size) available.

To simplify cloud installation, Driverless AI is provided as an AMI. To simplify local installation, Driverless AI is provided as a Docker image. 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).

Driverless AI supports HDFS and S3 access using either run-time options or by specifying a config.toml file. More information is available in the Driverless AI User Guide.

Driverless AI also supports basic, LDAP, and PAM authentication, which admins can configure via a config.toml file. More information is available in the Driverless AI User Guide.

The installation steps vary based on your platform. Refer to the following tables to find the right setup instructions for your environment. Note that each of these installation steps assumes that you have a license key for Driverless AI. For information on how to purchase a license key for Driverless AI, contact `sales@h2o.ai`.

Installation Tables by Environment

Use the following tables for Cloud, Server, and Desktop to find the right setup instructions for your environment. The **Refer to Section** column describes the section in the Driverless AI User Guide that contains the relevant instructions.

Cloud

Provider	Instance Type	Num GPUs	Suitable for	Refer to Section
NVIDIA GPU Cloud			Serious use	Install on NVIDIA GPU/DGX
AWS	p2.xlarge	1	Experimentation	Install on AWS
	p2.8xlarge	8	Serious use	
	p2.16xlarge	16	Serious use	
	p3.2xlarge	1	Experimentation	
	p3.8xlarge	4	Serious use	
	p3.16xlarge	8	Serious use	
	g3.4xlarge	1	Experimentation	
	g3.8xlarge	2	Experimentation	
	g3.16xlarge	4	Serious use	
Azure	Standard_NV6	1	Experimentation	Install on Azure
	Standard_NV12	2	Experimentation	
	Standard_NV24	4	Serious use	
	Standard_NC6	1	Experimentation	
	Standard_NC12	2	Experimentation	
	Standard_NC24	4	Serious use	
Google Compute with GPUs				Install on Google Compute with GPUs
Google Compute with CPUs				Install on Google Compute with CPUs

Server

Operating System	GPUs?	Min Mem	Refer to Section

NVIDIA DGX-1	Yes	128 GB	Install on NVIDIA GPU/DGX
Ubuntu 16.0.4	Yes	64 GB	Install on Ubuntu with GPUs
Ubuntu with CPUs	No	64 GB	Install on Ubuntu with CPUs
RHEL with GPUs	Yes	64 GB	Install on RHEL with GPUs
RHEL with CPUs	No	64 GB	Install on RHEL with CPUs
IBM Power (Minsky)	Yes	64 GB	Contact sales@h2o.ai

Desktop

Operating System	GPUs?	Min Mem	Suitable for	Refer to Section
NVIDIA DGX Station	Yes	64 GB	Serious use	Install on NVIDIA GPU/DGX
Mac OS X	No	16 GB	Experimentation	Install on Mac OS X
Windows 10 Pro	No	16 GB	Experimentation	Install on Windows 10 Pro
Linux	See **Server** table			

Running an Experiment

1. After Driverless AI is installed and started, open a browser (Chrome recommended) and navigate to `<driverless-ai-host-machine>: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:

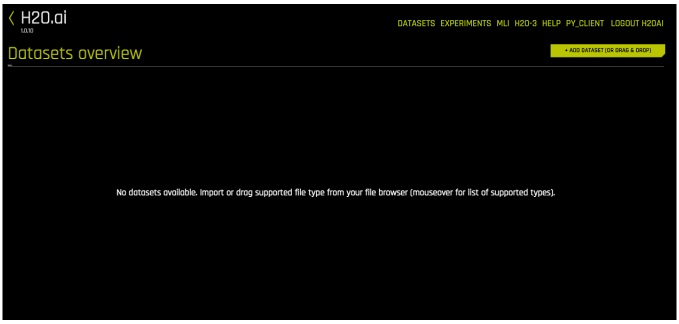
```
Username: h2oai
Password: h2oai
```

   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. Paste the License Key into the **License Key** entry field, and then 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.

5. The Home page appears, showing all datasets that have been imported. Note that the first time you log in, this list will be empty.


H2O.ai
1.0.10
DATASETS EXPERIMENTS MLI H2O-3 HELP PY_CLIENT LOGOUT H2OAI
+ ADD DATASET (OR DRAG & DROP)
Datasets overview

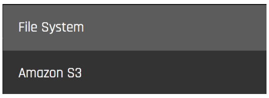
No datasets available. Import or drag supported file type from your file browser (mouseover for list of supported types).

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

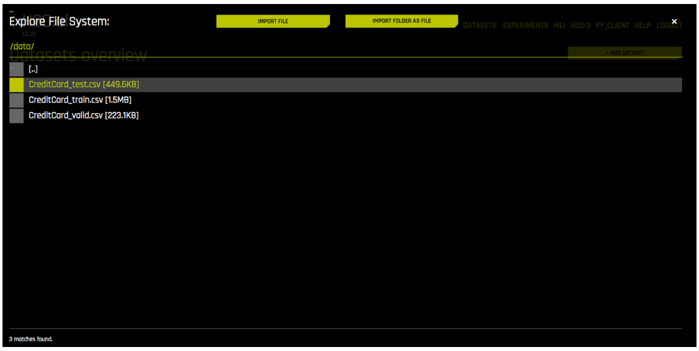
(a) Click the **Add Dataset** button. Note that if Driverless AI was started with data connectors enabled for HDFS and/or S3, then a dropdown will appear allowing you to specify where to begin browsing for the dataset. Refer to the Driverless AI User Guide for more information.



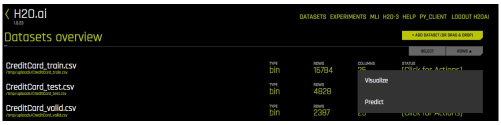
(b) In the **Explore File System** field, type the location for the dataset. Driverless AI autofills the browse line as type in the file location. When you locate the folder that includes your datasets, you can specify to import the folder or to import one or more files.

Notes:

- 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.


Explore File System:
IMPORT FILE
IMPORT FOLDER AS FILE
/data/
[..]
CreditCard_test.csv [449.6KB]
CreditCard_train.csv [1.5MB]
CreditCard_valid.csv [223.1KB]
3 matches found.

6. After importing your data, you can run an experiment by selecting **[Click for Actions]** button beside the dataset that you want to use. This opens a submenu that allows you to Visualize or Predict a dataset. (**Note**: You can delete an unused dataset by hovering over it and clicking the **X** button, and then confirming the delete. You cannot delete a dataset that was used in an active experiment. You have to delete the experiment first.) Click **Predict** to begin an experiment.



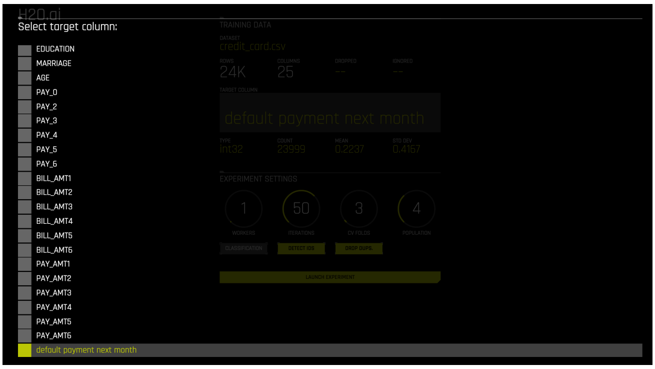
7. The Experiment Settings form displays and auto-fills with the selected dataset. Optionally specify a validation dataset and/or a test dataset.

   - 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).

8. Specify the target (response) column. Note that not all explanatory functionality will be available for multinomial 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 for numerical columns. If this is a regression problem, then the UI shows the dataset mean and standard deviation values. At this point, you can configure the following experiment settings.

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.

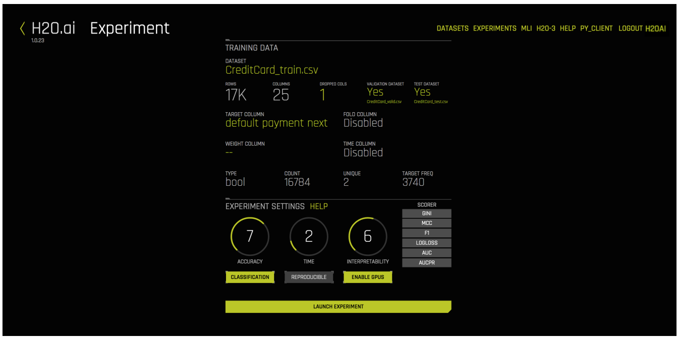
9. The next step is to set the parameters and settings for the experiment. (Refer to the Experiment Settings section that follows for more information about these settings.) You can set the parameters individually, or you

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.
- 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.
- Desired relative accuracy from 1 to 10
- Desired relative time from 1 to 10
- Desired relative interpretability from 1 to 10
- Specify the scorer to use for this experiment. If not selected, Driverless AI will select one based on the dataset and experiment. Available scorers include:
  - Regression: GINI, R2, MSE, RMSE, RMSLE, RMSPE, MAE, MAPE, SMAPE
  - Classification: GINI, MCC, F1, Logloss, AUC, AUCPR

Additional settings:

- If this is a classification problem, then click the **Classification** button. Note that Driverless AI determines the problem type based on the response column. Though not recommended, you can override this setting and specify whether this is a classification or regression problem.
- Click the **Reproducible** button to build this with a random seed.
- Specify whether to enable GPUs. (Note that this option is ignored on CPU-only systems.)



TRAINING DATA

DATASET
CreditCard_train.csv

ROWS	COLUMNS	DROPPED COLS	VALIDATION DATASET	TEST DATASET
17K	25	1	Yes CreditCard_valid.csv	Yes CreditCard_test.csv

TARGET COLUMN: default payment next
FOLD COLUMN: Disabled
WEIGHT COLUMN: --
TIME COLUMN: Disabled

TYPE	COUNT	UNIQUE	TARGET FREQ
bool	16784	2	3740

ACCURACY 7
TIME 2
INTERPRETABILITY 6

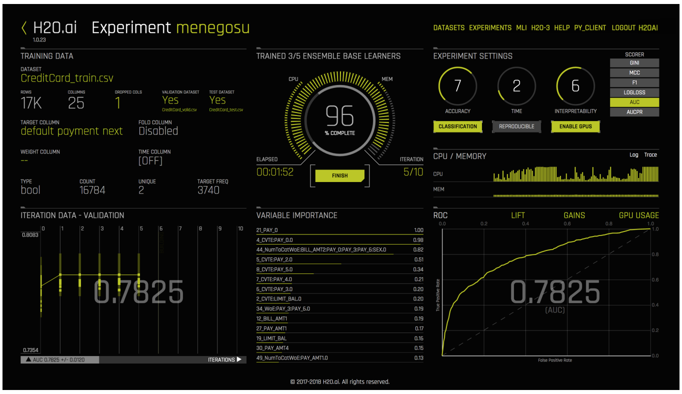
10. Click **Launch Experiment** to start the experiment.

    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 score (internal validation) for each cross validation fold along with any specified scorer value, the variable importance values, CPU/Memory information, and a toggle between an ROC curve, Lift chart, Gains chart, and GPU Usage information (if GPUs are available). Upon completion, an Experiment Summary section will populate in the lower right section. In this section, you can toggle between the Experiment Summary, ROC curve, Lift chart, and Gains chart.

    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.) Note that aborted experiments will not display on the Experiments page.



TRAINING DATA

DATASET: CreditCard_train.csv

ROWS: 17K COLUMNS: 25 DROPPED COLS: 1 VALIDATION DATASET: Yes (CreditCard_valid.csv) TEST DATASET: Yes (CreditCard_test.csv)

TARGET COLUMN: default payment next FOLD COLUMN: Disabled

WEIGHT COLUMN: -- TIME COLUMN: [OFF]

TYPE: bool COUNT: 16784 UNIQUE: 2 TARGET FREQ: 3740

ELAPSED: 00:01:52 ITERATION: 5/10

EXPERIMENT SETTINGS: 7 ACCURACY, 2 TIME, 6 INTERPRETABILITY

VARIABLE IMPORTANCE

21_PAY_0	1.00
4_CVTE:PAY_0.0	0.98
44_NumToCatWoE:BILL_AMT2:PAY_0:PAY_3:PAY_6:SEX.0	0.82
5_CVTE:PAY_2.0	0.51
8_CVTE:PAY_5.0	0.34
7_CVTE:PAY_4.0	0.21
6_CVTE:PAY_3.0	0.20
2_CVTE:LIMIT_BAL.0	0.20
34_WoE:PAY_3:PAY_6.0	0.19
12_BILL_AMT1	0.19
27_PAY_AMT1	0.17
19_LIMIT_BAL	0.16
30_PAY_AMT4	0.15
49_NumToCatWoE:PAY_AMT1.0	0.13

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.

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 >= 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 scoring of the test set predictions can use the weight).

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.

Time Column

Optional: Column that provides a time order, if applicable. Can improve model performance and model validation accuracy for problems where the target values are auto-correlated with respect to the ordering. Each observationś time stamp is used to order the observations in a causal way (generally, to avoid training on the future to predict the past).

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". If [AUTO] is selected, all string columns are tested for potential date/datetime content and considered as potential time columns. The natural row order of the training data is also considered in case no date/datetime columns are detected. If the data is (nearly) identically and independently distributed (i.i.d.), then no time column is needed. If [OFF] is selected, no time order is used for modeling, and data may be shuffled randomly (any potential temporal causality will be ignored). Note that a Time Column cannot be specified if a validation set is used.

Accuracy

The following table describes how the Accuracy value affects a Driverless AI experiment.

Accuracy	Max Rows	Ensemble Level	Target Transformation	Parameter Tuning Level	Num Individuals	Num Folds	Only First Fold Model	Distribution Check
1	10K	0	False	0	Auto	3	True	No
2	100K	0	False	0	Auto	3	True	No
3	500k	0	False	0	Auto	3	True	No
4	1M	0	False	0	Auto	3-4	True	No
5	2M	1	True	1	Auto	3-4	True	Yes
6	5M	1	True	1	Auto	3-5	True	Yes
7	10M	<=2	True	2	Auto	3-10	True	Yes
8	10M	<=2	True	2	Auto	4-10	if >= 5M rows	Yes
9	20M	<=3	True	3	Auto	4-10	if >= 5M rows	Yes
10	None	<=3	True	3	Auto	4-10	if >= 5M rows	Yes

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:** the maximum number of rows to use in model training.
  - For classification, stratified random sampling is performed
  - For regression, random sampling is performed
- **Ensemble Level:** The level of ensembling done for the final model
  - 0: single final model
  - 1: 2 4-fold final models ensembled together
  - 2: 5 5-fold final models ensembled together
  - 3: 8 5-fold final models ensembled together
- **Target Transformation:** Try target transformations and choose the transformation that has the best score.
  - Possible transformations: identity, unit_box, log, square, square root, inverse, Anscombe, logit, sigmoid
- **Parameter Tuning Level**: The level of parameter tuning done
  - 0: no parameter tuning
  - 1: 8 different parameter settings
  - 2: 16 different parameter settings

  - 3: 32 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.
- **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.

Time	Iterations	Early Stopping Rounds
1	1	None
2	10	5
3	30	5
4	40	5
5	50	10
6	100	10
7	150	15
8	200	20
9	300	30
10	500	50

Note: See the Accuracy table for cases when not based upon time.

Interpretability

Interpretability

Interpretability	Ensemble Level	Monotonicity Constraints
<= 5	<= 3	Disabled
>= 6	<= 2	Disabled
>= 7	<= 2	Enabled
>= 8	<= 1	Enabled
10	0	Enabled

Interpretability	Transformers**
<= 5	All
6	Interpretability#5 - [TruncSvdNum, ClusterDist]
7	Interpretability#6 - [ClusterIDTargetEncodeMulti, ClusterIDTargetEncodeSingle]
8	Interpretability#7 - [NumToCatTargetEncodeSingle, NumToCatTargetEncodeMulti, Frequent]
9	Interpretability#8 - [NumToCatWeightOfEvidence, NumToCatWoE, NumCatTargetEncodeMulti, NumCatTargetEncodeSingle]
10	Interpretability#9 - [BulkInteractions, WeightOfEvidence, CvCatNumEncode, NumToCatWeightOfEvidenceMonotonic]

**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.

Interpretability	Target Transformer
<=10	TargetTransformer_identity
<=10	TargetTransformer_unit_box
<=10	TargetTransformer_log
<= 9	TargetTransformer_square
<= 9	TargetTransformer_sqrt
<= 6	TargetTransformer_logit
<= 6	TargetTransformer_sigmoid
<= 5	TargetTransformer_Anscombe
<= 4	TargetTransformer_inverse

- **Monotonicity Constraints:**

  If 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.

- **Date Types Detected**

  – categorical

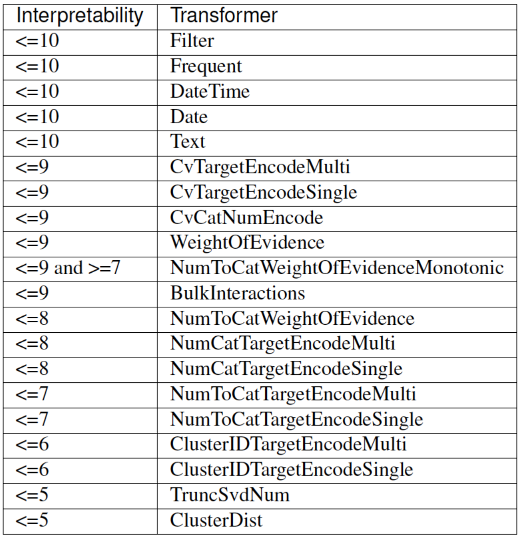
  – date

  – datetime

  – numeric

  – text

- **Transformers used on raw features to generate new features**:

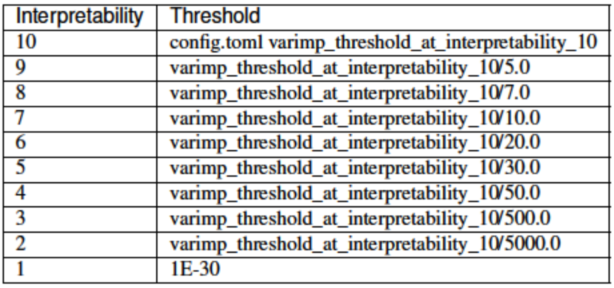


Interpretability	Transformer
<=10	Filter
<=10	Frequent
<=10	DateTime
<=10	Date
<=10	Text
<=9	CvTargetEncodeMulti
<=9	CvTargetEncodeSingle
<=9	CvCatNumEncode
<=9	WeightOfEvidence
<=9 and >=7	NumToCatWeightOfEvidenceMonotonic
<=9	BulkInteractions
<=8	NumToCatWeightOfEvidence
<=8	NumCatTargetEncodeMulti
<=8	NumCatTargetEncodeSingle
<=7	NumToCatTargetEncodeMulti
<=7	NumToCatTargetEncodeSingle
<=6	ClusterIDTargetEncodeMulti
<=6	ClusterIDTargetEncodeSingle
<=5	TruncSvdNum
<=5	ClusterDist

** Default N-way interactions are up to 8-way except:

- BulkInteractions are always 2-way.
- Frequent is set to 1-way if interpretability=10.
- Interactions are minimal-way (for example, 1-way for CvTargetEncoding)

- **Variable Importance Threshold in Below which Features are Removed**



Interpretability	Threshold
10	config.toml varimp_threshold_at_interpretability_10
9	varimp_threshold_at_interpretability_10/5.0
8	varimp_threshold_at_interpretability_10/7.0
7	varimp_threshold_at_interpretability_10/10.0
6	varimp_threshold_at_interpretability_10/20.0
5	varimp_threshold_at_interpretability_10/30.0
4	varimp_threshold_at_interpretability_10/50.0
3	varimp_threshold_at_interpretability_10/500.0
2	varimp_threshold_at_interpretability_10/5000.0
1	1E-30

Interpreting a Model

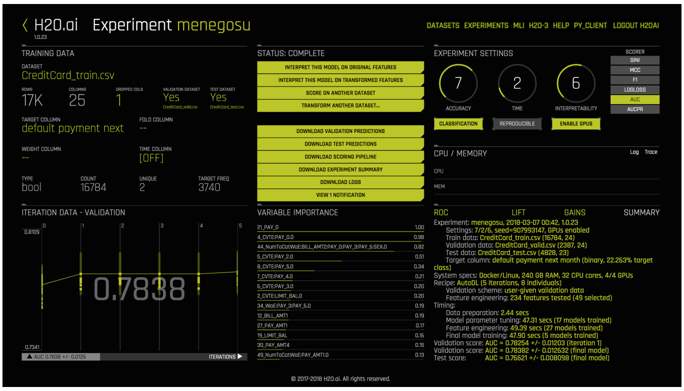
There are two methods you can use for interpreting models:

- Using the **Interpret this Model** button on a completed experiment page to interpret a Driverless AI model.
- Using the **MLI** link in the upper right corner of the UI to interpret either a Driverless AI model or an external model.

Interpret this Model button

After the status changes from RUNNING to COMPLETE, the UI provides you with several options:

- Interpret this Model on Original Features
- Interpret this Model on Transformed Features
- Score on Another Dataset (Refer to the Score on Another Dataset section.)
- Transform Another Dataset (Refer to the Transform Another Dataset section.)
- Download (Holdout) Training Predictions (in csv format, available if a validation set was NOT used)
- Download Validation Predictions (in csv format, available if a validation set was used)
- Download Test Predictions (in csv format, available if a test dataset is used)
- Download Scoring Pipeline (A standalone Python scoring pipeline for H2O Driverless AI. Refer to The Scoring Pipelines section.)
- Download Experiment Summary (a zip file containing a summary of the experiment and the features along with their relative importance)
- Download Logs
- View Notifications/Warnings (if any existed)


H2O.ai
1.0.23
Experiment menegosu
DATASETS EXPERIMENTS MLI H2O-3 HELP PY_CLIENT LOGOUT H2OAI
STATUS: COMPLETE
INTERPRET THIS MODEL ON ORIGINAL FEATURES
INTERPRET THIS MODEL ON TRANSFORMED FEATURES
SCORE ON ANOTHER DATASET
TRANSFORM ANOTHER DATASET...
DOWNLOAD VALIDATION PREDICTIONS
DOWNLOAD TEST PREDICTIONS
DOWNLOAD SCORING PIPELINE
DOWNLOAD EXPERIMENT SUMMARY
DOWNLOAD LOGS
VIEW 1 NOTIFICATION
SCORER
GINI
MCC
F1
LOGLOSS
AUC
AUCPR
CLASSIFICATION
REPRODUCIBLE
ENABLE GPUS
CPU / MEMORY
Log
Trace
CPU
MEM
ITERATION DATA - VALIDATION
0.8109
0.7341
0.7838
AUC 0.7838 +/- 0.0126
ITERATIONS
ROC
LIFT
GAINS
SUMMARY
© 2017-2018 H2O.ai. All rights reserved.

TRAINING DATA

DATASET	ROWS	COLUMNS	DROPPED COLS	VALIDATION DATASET	TEST DATASET
CreditCard_train.csv	17K	25	1	Yes (CreditCard_valid.csv)	Yes (CreditCard_test.csv)

TARGET COLUMN	FOLD COLUMN	WEIGHT COLUMN	TIME COLUMN
default payment next	--	--	[OFF]

TYPE	COUNT	UNIQUE	TARGET FREQ
bool	16784	2	3740

EXPERIMENT SETTINGS

ACCURACY	TIME	INTERPRETABILITY
7	2	6

VARIABLE IMPORTANCE

21_PAY_0	1.00
4_CVTE:PAY_0.0	0.98
44_NumToCatWoE:BILL_AMT2:PAY_0:PAY_3:PAY_6:SEX.0	0.82
5_CVTE:PAY_2.0	0.51
8_CVTE:PAY_5.0	0.34
7_CVTE:PAY_4.0	0.21
6_CVTE:PAY_3.0	0.20
2_CVTE:LIMIT_BAL.0	0.20
34_WoE:PAY_3:PAY_6.0	0.19
12_BILL_AMT1	0.19
27_PAY_AMT1	0.17
19_LIMIT_BAL	0.16
30_PAY_AMT4	0.15
49_NumToCatWoE:PAY_AMT1.0	0.13

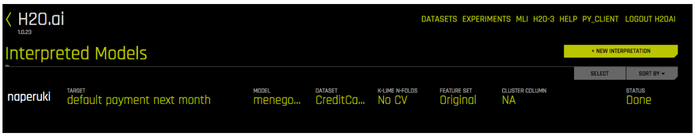
```
Experiment: menegosu, 2018-03-07 00:42, 1.0.23
    Settings: 7/2/6, seed=907993147, GPUs enabled
    Train data: CreditCard_train.csv (16784, 24)
    Validation data: CreditCard_valid.csv (2387, 24)
    Test data: CreditCard_test.csv (4828, 23)
    Target column: default payment next month (binary, 22.263% target
class)
System specs: Docker/Linux, 240 GB RAM, 32 CPU cores, 4/4 GPUs
Recipe: AutoDL (5 iterations, 8 individuals)
    Validation scheme: user-given validation data
    Feature engineering: 234 features tested (49 selected)
Timing:
    Data preparation: 2.44 secs
    Model parameter tuning: 47.31 secs (17 models trained)
    Feature engineering: 49.39 secs (27 models trained)
    Final model training: 47.90 secs (5 models trained)
Validation score: AUC = 0.78254 +/- 0.01203 (iteration 1)
Validation score: AUC = 0.78382 +/- 0.012632 (final model)
Test score:       AUC = 0.76621 +/- 0.008098 (final model)
```

Click one of the **Interpret this Model** buttons to launch the Model Interpretation page. This page provides several visual explanations of the trained Driverless AI model and its results.

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 one of the **Interpret This Model** buttons 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.

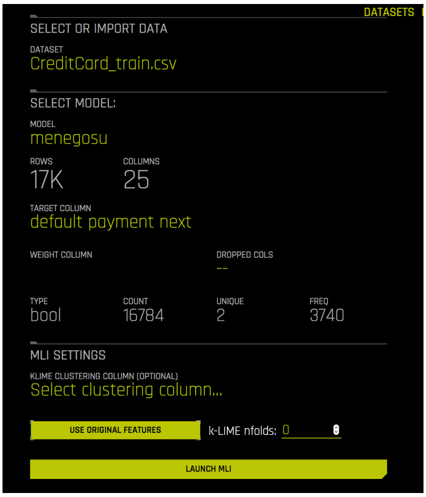


	TARGET	MODEL	DATASET	K-LIME N-FOLDS	FEATURE SET	CLUSTER COLUMN	STATUS
naperuki	default payment next month	menego...	CreditCa...	No CV	Original	NA	Done

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.
5. Specify the column of the target variable (the column of actuals for MLI).
6. Optionally specify weight and dropped columns.
7. Optionally specify a clustering column and whether to use the original features.

8. Optionally specify the number of cross-validation folds to use in k-LIME. This defaults to 0, and the maximum value is 10.

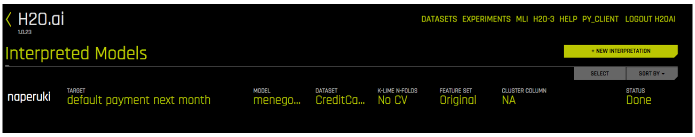
9. Click the **Launch MLI** button.


SELECT OR IMPORT DATA
DATASET CreditCard_train.csv
SELECT MODEL:
MODEL menegosu
ROWS 17K COLUMNS 25
TARGET COLUMN default payment next
WEIGHT COLUMN
DROPPED COLS --
TYPE bool COUNT 16784 UNIQUE 2 FREQ 3740
MLI SETTINGS
KLIME CLUSTERING COLUMN (OPTIONAL) Select clustering column...
k-LIME nfolds: 0

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.


Interpreted Models
naperuki
TARGET default payment next month
MODEL menego...
DATASET CreditCa...
K-LIME N-FOLDS No CV
FEATURE SET Original
CLUSTER COLUMN NA
STATUS Done

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 using Python is available in the Run Model Interpretation on External Model Predictions section.

Note: When running interpretations on an external model, leave the **Select Model** option empty. That option is for selecting a Driverless AI model.

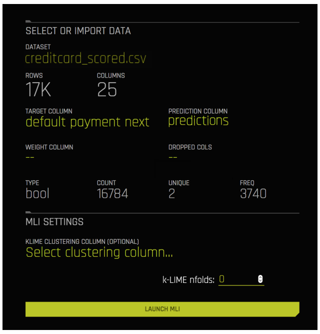
4. Specify a Target Column (actuals) and the Prediction Column (scores from the model).

5. Optionally specify weight and dropped columns.

6. Optionally specify a clustering column.

7. Optionally specify the number of cross-validation folds to use in k-LIME. This defaults to 0, and the maximum value is 10.

8. Click the **Launch MLI** button.



SELECT OR IMPORT DATA

DATASET
creditcard_scored.csv

ROWS	COLUMNS
17K	25

TARGET COLUMN	PREDICTION COLUMN
default payment next	predictions

WEIGHT COLUMN	DROPPED COLS
--	--

TYPE	COUNT	UNIQUE	FREQ
bool	16784	2	3740

MLI SETTINGS

KLIME CLUSTERING COLUMN (OPTIONAL)
Select clustering column...

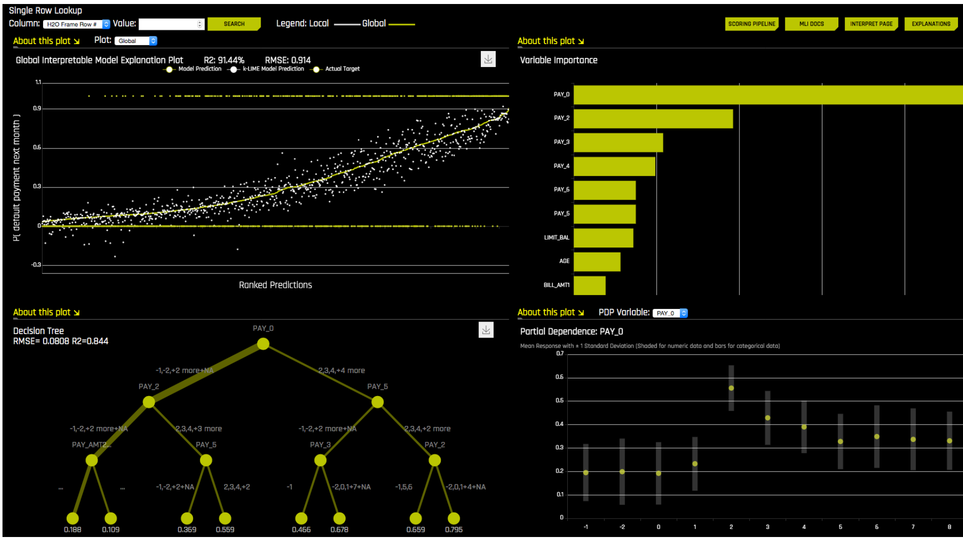
k-LIME nfolds: 0

The Model Interpretation Page

The Model Interpretation page includes the following information:

- Global interpretable model explanation plot
- Variable importance
- Decision tree surrogate model
- Partial dependence and individual conditional expectation plots

Each of these plots and techniques provide different types of insights and explanations regarding a model and its results.



K-LIME

The K-LIME Technique

K-LIME is a variant of the LIME technique proposed by Ribeiro at al [9]. *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. 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 variable 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:

debt_to_income_ ratio	credit_ score	savings_acct_ balance	observed_ default	H2OAI_predicted_ default	k-LIME_predicted_ default
30	600	1000	1	0.85	0.9

And the local linear model:

$$y_{K\text{-LIME}} = 0.1 + 0.01 * debt\_to\_income\_ratio + 0.0005 * credit\_score + 0.0002 * savings\_account\_balance$$

It can be seen that the local linear contributions for each variable are:

- *debt_to_income_ratio*: 0.01 * 30 = 0.3
- *credit_score*: 0.0005 * 600 = 0.3
- *savings_acct_balance*: 0.0002 * 1000 = 0.2

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*.

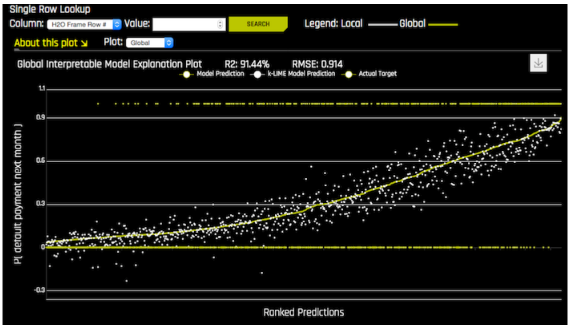
The local linear model intercept and the products of each coefficient and corresponding value sum to the *K*-LIME prediction. Moreover it can be seen that these linear explanations are reasonably representative of the nonlinear model's behavior for this individual because the *K*-LIME predictions are 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 variable 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 Global Interpretable Model Explanation Plot

This plot is in the upper-left quadrant of the UI. It shows Driverless AI model predictions and *K*-LIME model predictions in sorted order by the Driverless AI model predictions. This graph is interactive. Hover over the **Model Prediction**, **K-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 *K*-LIME reason codes for that value. By default, this plot shows information for the global *K*-LIME model, but you can change the plot view to show local results from a specific cluster. The *K*-LIME plot also provides a visual indication of the linearity of the Driverless AI model and the trustworthiness of the *K*-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 *K*-LIME local linear models.



Global and Local Variable Importance

Variable importance measures the effect that a variable has on the predictions of a model. Global and local variable importance values enable increased transparency in the Driverless AI model and enable validating and debugging of the Driverless AI model by comparing global model behavior to the local model behavior, and by comparing to global and local variable importance to known standards, domain knowledge, and reasonable expectations.

Global Variable Importance Technique

Global variable importance measures the overall impact of an input variable on the Driverless AI model predictions while taking nonlinearity and interactions into consideration. Global variable importance values give an indication of the magnitude of a variable's contribution to model predictions for all rows. Unlike regression parameters, they are often unsigned and typically not directly related to the numerical predictions of the model. The reported global variable importance values are calculated by aggregating the improvement in the split-criterion for a variable across all the trees in an ensemble. The aggregated feature importance values are then scaled between 0 and 1, such that the most important feature has an importance value of 1.

Local Variable Importance Technique

Local variable 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 variable importance values reported here are based on a variant of the leave-one-covariate-out (LOCO) method (Lei et al, 2017 [8]).

In the LOCO-variant method, each local variable 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 variable 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 variable importance values.

Given the row of input data with its corresponding Driverless AI and K-LIME predictions:

debt_to_income_ratio	credit_score	savings_acct_balance	observed_default	H2OAI_predicted_default	k-LIME_predicted_default
30	600	1000	1	0.85	0.9

Taking the Driverless AI model as F(**X**), LOCO-variant variable importance values are calculated as follows.

First, the modified predictions are calculated:

$$F_{debt\_to\_income\_ratio} = F(NA, 600, 1000) = 0.99$$

$$F_{credit\_score} = F(30, NA, 1000) = 0.73$$

$$F_{savings\_acct\_balance} = F(30, 600, NA) = 0.82$$

Second, the original prediction is subtracted from each modified prediction to generate the unscaled local variable importance values:

$$\text{LOCO}_{debt\_to\_income\_ratio} = F_{debt\_to\_income\_ratio} - 0.85 = 0.99 - 0.85 = 0.14$$

$$\text{LOCO}_{credit\_score} = F_{credit\_score} - 0.85 = 0.73 - 0.85 = -0.12$$

$$\text{LOCO}_{savings\_acct\_balance} = F_{savings\_acct\_balance} - 0.85 = 0.82 - 0.85 = -0.03$$

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.

$$\text{Scaled}(\text{LOCO}_{debt\_to\_income\_ratio}) = \text{Abs}(\text{LOCO}_{debt\_to\_income\_ratio}/0.14) = 1$$

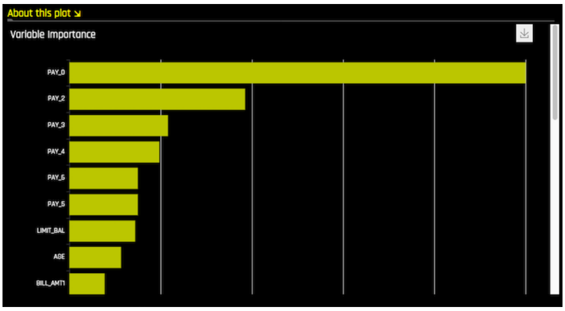
$$\text{Scaled}(\text{LOCO}_{credit\_score}) = \text{Abs}(\text{LOCO}_{credit\_score}/0.14) = 0.86$$

$$\text{Scaled}(\text{LOCO}_{savings\_acct\_balance}) = \text{Abs}(\text{LOCO}_{savings\_acct\_balance}/0.14) = 0.21$$

One drawback to these LOCO-variant variable importance values is, unlike K-LIME, it is difficult to generate a mathematical error rate to indicate when LOCO values may be questionable.

The Variable Importance Plot

The upper-right quadrant of the Model Interpretation page shows the scaled global variable importance values for the features in the model. Hover over each bar in the graph to view the scaled global importance value for that feature. When a specific row is selected, scaled local variable importance values are shown alongside scaled global variable importance values for comparison.



Decision Tree Surrogate Model

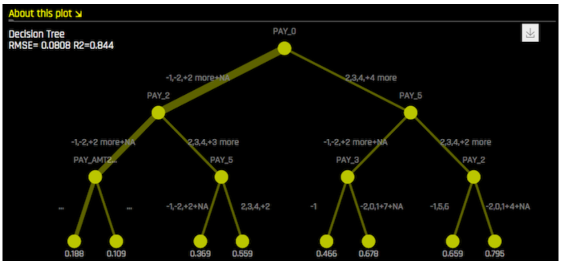
The Decision Tree Surrogate Model Technique

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, [2]). 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 Surrogate Model Plot

The lower-left quadrant shows a decision tree surrogate for the generated model. 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.



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 all, 2001 [3]). 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 [4]). 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:

debt_to_income_ ratio	credit_ score	savings_acct_ balance	observed_ default	H2OAI_predicted_ default	k-LIME_predicted_ default
30	600	1000	1	0.85	0.9

Taking the Driverless AI model as F($\mathbf{X}$), 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:

$$\text{ICE}_{credit\_score,500} = F(30, 500, 1000)$$

$$\text{ICE}_{credit\_score,530} = F(30, 530, 1000)$$

$$\text{ICE}_{credit\_score,560} = F(30, 560, 1000)$$

...

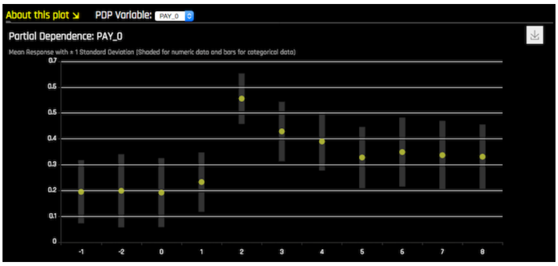
$$\text{ICE}_{credit\_score,800} = F(30, 800, 1000)$$

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 and Individual Conditional Expectation 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.

The lower-right quadrant shows the partial dependence for a selected variable 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. By default, this graph shows the partial dependence values for the top feature. Change this view by selecting a different feature in the feature drop-down. Note that this graph is available for the top five features.



General Considerations

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 [6]), 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 [9]).

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 (Brieman, 2001 [1]). 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.

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 variable 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 variable 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.

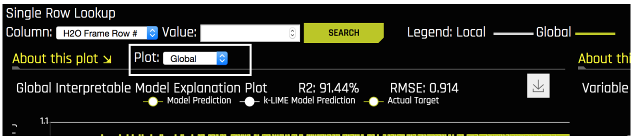
Viewing Explanations

Note: Not all explanatory functionality is available for multinomial classification scenarios.

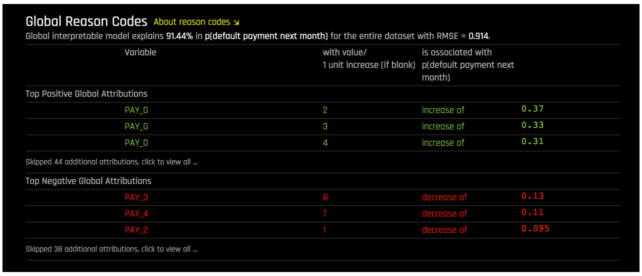
Driverless AI provides easy-to-read explanations for a completed model. You can view these by clicking the **Explanations** button in the upper-right corner of the Model Interpretation page. Note that this button is only available for completed experiments. Click **Close** when you are done to return to the Model Interpretations page.

The UI allows you to view global, cluster-specific, and local reason codes.

- **Global Reason Codes**: To view global reason codes, select the Global plot from the **Plot** dropdown.


Column: H2O Frame Row #
Plot: Global
Global Interpretable Model Explanation Plot R2: 91.44% RMSE: 0.914

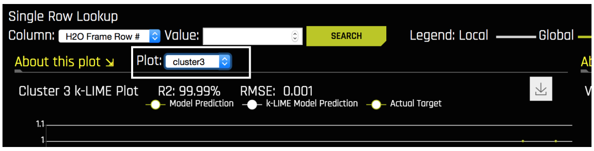
With Global selected, click the **Explanations** button in the upper-right corner.


Global Reason Codes

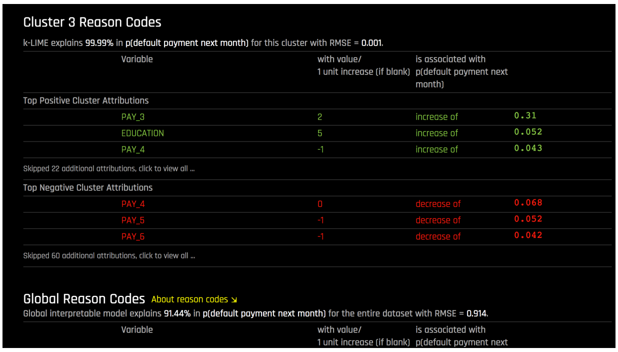
Global interpretable model explains 91.44% in p(default payment next month) for the entire dataset with RMSE = 0.914.

Variable	with value/ 1 unit increase (if blank)	is associated with p(default payment next month)	
Top Positive Global Attributions			
PAY_0	2	increase of	0.37
PAY_0	3	increase of	0.33
PAY_0	4	increase of	0.31
Skipped 44 additional attributions, click to view all ...			
Top Negative Global Attributions			
PAY_3	8	decrease of	0.13
PAY_4	7	decrease of	0.11
PAY_2	1	decrease of	0.095
Skipped 38 additional attributions, click to view all ...			

- **Cluster Reason Codes**: To view reason codes for a specific cluster, select a cluster from the **Plot** dropdown.



With a cluster selected, click the **Explanations** button in the upper-right corner.



Cluster 3 Reason Codes

k-LIME explains 99.99% in p(default payment next month) for this cluster with RMSE = 0.001.

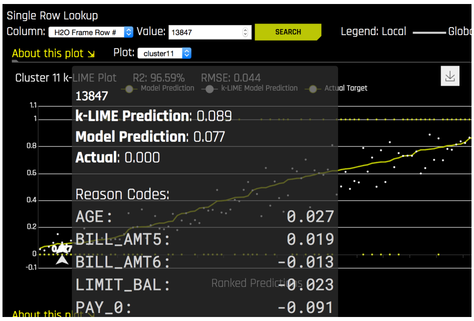
Variable	with value/ 1 unit increase (if blank)	is associated with p(default payment next month)	
Top Positive Cluster Attributions			
PAY_3	2	increase of	0.31
EDUCATION	5	increase of	0.052
PAY_4	-1	increase of	0.043
Skipped 22 additional attributions, click to view all ...			
Top Negative Cluster Attributions			
PAY_4	0	decrease of	0.068
PAY_5	-1	decrease of	0.052
PAY_6	-1	decrease of	0.042
Skipped 60 additional attributions, click to view all ...			

Global Reason Codes About reason codes

Global interpretable model explains 91.44% in p(default payment next month) for the entire dataset with RMSE = 0.914.

Variable	with value/ 1 unit increase (if blank)	is associated with p(default payment next	

- **Local Reason Codes**: To view local reason codes, select a point on the graph or type a value in the Value field.



13847

k-LIME Prediction: 0.089

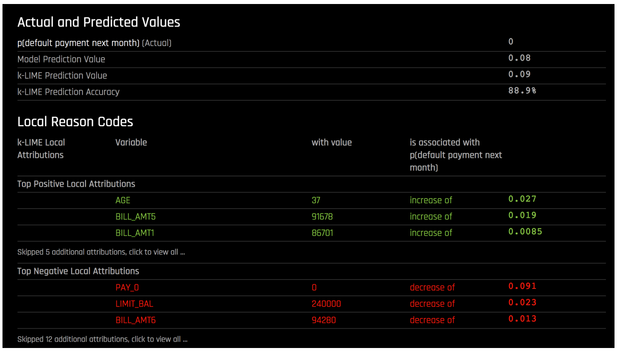
Model Prediction: 0.077

Actual: 0.000

Reason Codes:

```
AGE:          0.027
BILL_AMT5:    0.019
BILL_AMT6:   -0.013
LIMIT_BAL:   -0.023
PAY_0:       -0.091
```

With a value selected, click the **Explanations** button in the upper-right corner.



Actual and Predicted Values

p(default payment next month) (Actual)	0
Model Prediction Value	0.08
k-LIME Prediction Value	0.09
k-LIME Prediction Accuracy	88.9%

Local Reason Codes

k-LIME Local Attributions	Variable	with value	is associated with p(default payment next month)	
Top Positive Local Attributions				
	AGE	37	increase of	0.027
	BILL_AMT5	91678	increase of	0.019
	BILL_AMT1	86701	increase of	0.0085
Skipped 5 additional attributions, click to view all ...				
Top Negative Local Attributions				
	PAY_0	0	decrease of	0.091
	LIMIT_BAL	240000	decrease of	0.023
	BILL_AMT6	94280	decrease of	0.013
Skipped 12 additional attributions, click to view all ...				

Score on Another Dataset

After you generate a model, you can use that model to make predictions on another dataset.

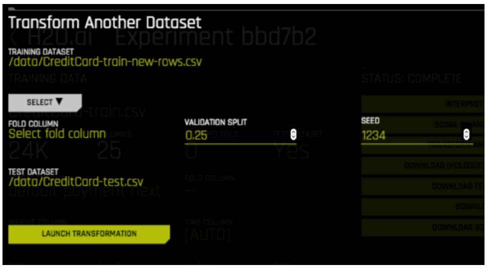
1. Click the **Experiments** link in the top menu and select the experiment that you want to use.
2. On the Experiments page, click the **Score on Another Dataset** button.
3. Locate the new dataset that you want to score on. Note that this new dataset must include the same columns as the dataset used 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.

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 About 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. The new rows will then get transformed into the original dataset.

Follow these steps to transform another dataset. Note that this assumes the new dataset has been added to Driverless AI already.

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 percent) and the seed (defaults to 1234).
4. Optionally specify a test dataset. If specified, then the output also include the final test dataset for final scoring.
5. Click **Launch Transformation**.


Transform Another Dataset

TRAINING DATASET
/data/CreditCard-train-new-rows.csv

SELECT ▼

FOLD COLUMN
Select fold column

VALIDATION SPLIT
0.25

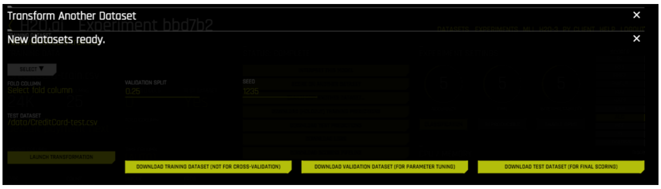
SEED
1234

TEST DATASET
/data/CreditCard-test.csv

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.


Transform Another Dataset

New datasets ready.

SELECT ▼

FOLD COLUMN
Select fold column

VALIDATION SPLIT
0.25

SEED
1235

TEST DATASET
/data/CreditCard-test.csv

LAUNCH TRANSFORMATION

DOWNLOAD TRAINING DATASET (NOT FOR CROSS-VALIDATION)

DOWNLOAD VALIDATION DATASET (FOR PARAMETER TUNING)

DOWNLOAD TEST DATASET (FOR FINAL SCORING)

The Scoring Pipelines

Scoring Pipelines are available for productionizing models and for obtaining reason codes on interpreted models for a given row of data.

Driverless AI Standalone Scoring Pipeline

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).

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 server.py).
- **run_http_server.sh**: Runs HTTP scoring service (runs 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).

Prerequisites

The following are required in order to run the downloaded scoring pipeline.

- Linux x86_64 environment

- Python 3.6 (Note that Anaconda Python 3.6 distribution is not supported at this point in time.)
- libopenblas-dev (required for H2O4GPU)
- Apache Thrift (to run the TCP scoring service)
- Internet access to download and install packages. Note that depending on your environment, you may also need to set up proxy.

The scoring pipeline has been tested on Ubuntu 16.04 and on 16.10+. Examples of how to install these prerequisites are below:

Installing Python 3.6 and OpenBlas Ubuntu 16.10+

```
$ sudo apt install python3.6 python3.6-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv libopenblas-dev
```

Installing Python 3.6 and OpenBLAS on Ubuntu 16.4

```
$ 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 libopenblas-dev
```

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
```

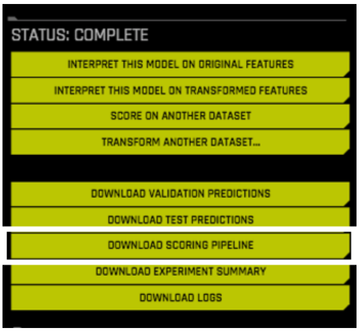
Run the following to refresh the runtime shared after installing Thrift:

```
$ sudo ldconfig /usr/local/lib
```

Quickstart

Before running these examples, be sure that the 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.


STATUS: COMPLETE
INTERPRET THIS MODEL ON ORIGINAL FEATURES
INTERPRET THIS MODEL ON TRANSFORMED FEATURES
SCORE ON ANOTHER DATASET
TRANSFORM ANOTHER DATASET...
DOWNLOAD VALIDATION PREDICTIONS
DOWNLOAD TEST PREDICTIONS
DOWNLOAD SCORING PIPELINE
DOWNLOAD EXPERIMENT SUMMARY
DOWNLOAD LOGS

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 x86_64 and Python 3.6.)

```
$ bash run_example.sh
```

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 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 run_tcp_server.sh
$ bash run_tcp_client.sh
```

Note: 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 at the end of this chapter to see how to set up and test the scoring module using a cleanroom Ubuntu 16.04 virtual machine.

The Scoring Module

The 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 = 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):

- 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:

```
# See 'run_example.sh' for complete example.
$ virtualenv -p python3.6 env
$ source env/bin/activate
$ pip install -r requirements.txt
$ python example.py
```

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).

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.

```
# 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(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();
    }
  }
}
```

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:

```
# 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",
  "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:

```
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'])
```

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 software-properties-common zip libopenblas-dev
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
      python-virtualenv python3-virtualenv

# end of bootstrap.sh

----- Vagrantfile -----

# -*- mode: ruby -*-
# vi: set ft=ruby :

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 are 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/
bash run_example.sh
```

At this point, you should see scores printed out on the terminal. If not, contact us at support@h2o.ai.

Driverless AI MLI Standalone 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.

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).

MLI 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.)
- **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.
- **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).

Prerequisites

- The scoring module and scoring service are supported only on Linux x86_64 with Python 3.6.
- Apache Thrift (to run the scoring service in TCP mode)
- The scoring module and scoring service needs access to Internet to download and install packages. Depending on your environment, you may also need to set up proxy.

Installing Python 3.6

Installing Python3.6 on Ubuntu 16.10+:

```
$ sudo apt install python3.6 python3.6-dev python3-pip python3-dev \
    python-virtualenv python3-virtualenv
```

Installing Python3.6 on Ubuntu 16.04

```
$ 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 the Thrift Compiler

Refer to Thrift documentation at https://thrift.apache.org/docs/BuildingFromSource for more information.

```
$ 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
```

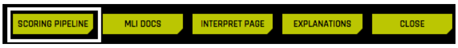
Run the following to refresh the runtime shared after installing Thrift.

```
$ sudo ldconfig /usr/local/lib
```

Quickstart

Before running the quickstart examples, be sure that the MLI Scoring Package is already downloaded and unzipped.

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 x86_64 and Python 3.6.)

```
$ bash run_example.sh
```

Run the TCP scoring server example. Use two terminal windows. (This requires Linux x86_64, Python 3.6 and Thrift.)

```
$ bash run_tcp_server.sh
$ bash run_tcp_client.sh
```

Run the HTTP scoring server example. Use two terminal windows. (This requires Linux x86_64, Python 3.6 and Thrift.)

```
$ bash run_http_server.sh
$ bash run_http_client.sh
```

MLI 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 KLime 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
```

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) or in batch mode (multiple rows at a time).

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.

```
----- 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();
    }
  }
}
```

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:

```
----- 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:

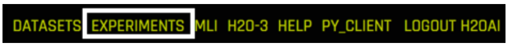
```
----- See 'run_http_client.sh' for complete example. -----
$ curl http://localhost:9090/rpc \
  --header "Content-Type: application/json" \
  --data @- <<EOF
 {
  "id": 1,
  "method": "score_reason_codes",
  "params": {
    "row": [ 7.486, 3.277, 4.755, 2.354 ]
  }
 }
EOF
```

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:

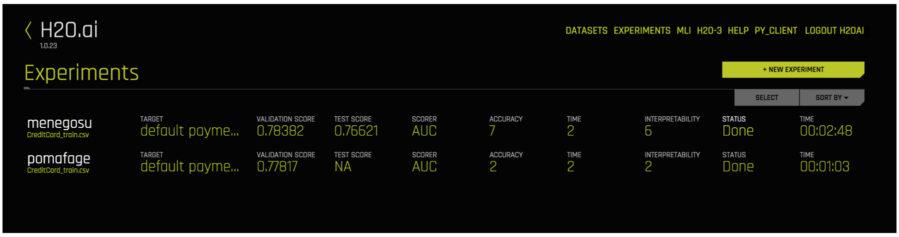
```
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score_reason_codes', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()['result'])
```

Viewing Experiments

The upper-right corner of the Driverless AI UI includes an **Experiments** link.



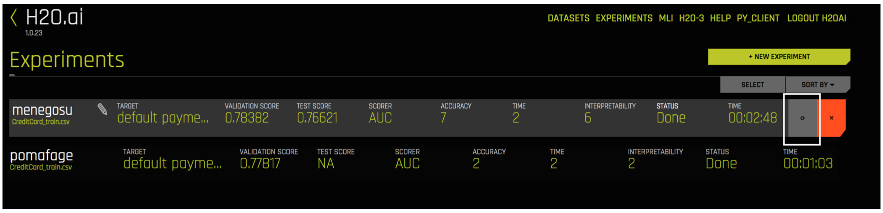
Click this link to open the Experiments page. From this page, you can rename an experiment, select and view previous experiments, and you can begin a new experiment.



	TARGET	VALIDATION SCORE	TEST SCORE	SCORER	ACCURACY	TIME	INTERPRETABILITY	STATUS	TIME
menegosu CreditCard_train.csv	default payme...	0.78382	0.76621	AUC	7	2	6	Done	00:02:48
pomafage CreditCard_train.csv	default payme...	0.77817	NA	AUC	2	2	2	Done	00:01:03

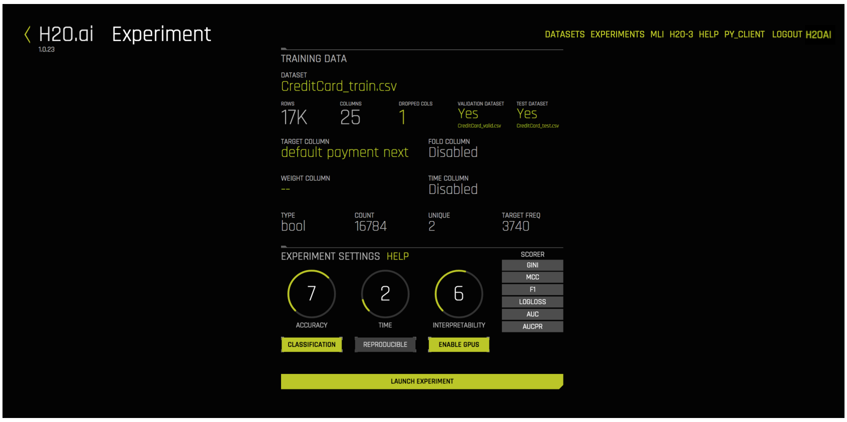
Rerunning Experiments

To rerun an experiment, or run a new experiment using an existing experiment's settings, hover over the experiment that you want to use. A "rerun" icon displays. Clicking this icon opens the selected experiment's settings.



	TARGET	VALIDATION SCORE	TEST SCORE	SCORER	ACCURACY	TIME	INTERPRETABILITY	STATUS	TIME
menegosu CreditCard_train.csv	default payme...	0.78382	0.76621	AUC	7	2	6	Done	00:02:48
pomafage CreditCard_train.csv	default payme...	0.77817	NA	AUC	2	2	2	Done	00:01:03

From the settings page, you can rerun the experiment using the original settings, or you can specify to use new data and/or specify different experiment settings.



TRAINING DATA

DATASET
CreditCard_train.csv

ROWS	COLUMNS	DROPPED COLS	VALIDATION DATASET	TEST DATASET
17K	25	1	Yes (CreditCard_valid.csv)	Yes (CreditCard_test.csv)

TARGET COLUMN: default payment next
FOLD COLUMN: Disabled
WEIGHT COLUMN: --
TIME COLUMN: Disabled

TYPE	COUNT	UNIQUE	TARGET FREQ
bool	16784	2	3740

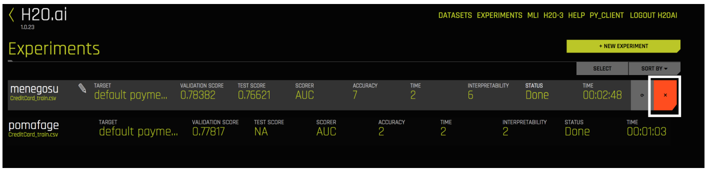
EXPERIMENT SETTINGS HELP

ACCURACY 7, TIME 2, INTERPRETABILITY 6

SCORER: GINI, MCC, F1, LOGLOSS, AUC, AUCPR

Deleting Experiments

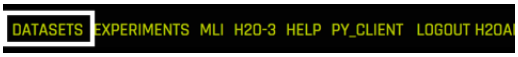
To delete an experiment, hover over the experiment that you want to delete. An "X" option displays. Click this to delete the experiment. A confirmation message will display asking you to confirm the delete. Click **OK** to delete the experiment or **Cancel** to return to the experiments page without deleting.



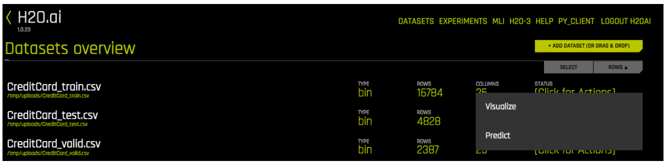
	TARGET	VALIDATION SCORE	TEST SCORE	SCORER	ACCURACY	TIME	INTERPRETABILITY	STATUS	TIME
menegosu (CreditCard_train.csv)	default payme...	0.78382	0.76621	AUC	7	2	6	Done	00:02:48
pomafoge (CreditCard_train.csv)	default payme...	0.77817	NA	AUC	2	2	2	Done	00:01:03

Visualizing Datasets

While viewing experiments, click the **Datasets** link in the upper-right corner.



The Datasets Overview page shows a list of the datasets that you've imported. Select the **[Click for Actions]** button beside the dataset that you want to view and then click **Visualize** from the submenu that appears.


H2O.ai
1.0.23
DATASETS EXPERIMENTS MLI H2O-3 HELP PY_CLIENT LOGOUT H2OAI
+ ADD DATASET (OR DRAG & DROP)
SELECT
ROWS ▲
Visualize
Predict

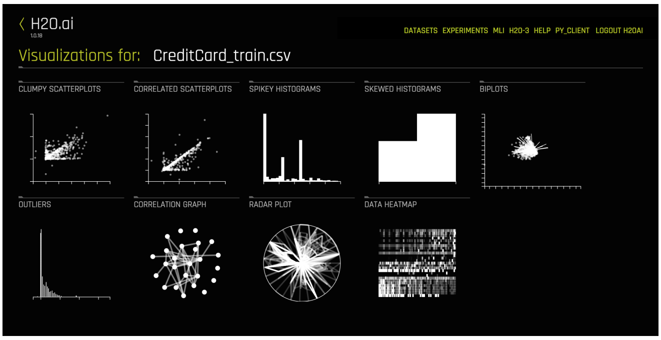
Datasets overview

CreditCard_train.csv
/tmp/uploads/CreditCard_train.csv
TYPE bin ROWS 16784 COLUMNS 25 STATUS [Click for Actions]

CreditCard_test.csv
/tmp/uploads/CreditCard_test.csv
TYPE bin ROWS 4828

CreditCard_valid.csv
/tmp/uploads/CreditCard_valid.csv
TYPE bin ROWS 2387 25 [Click for Actions]

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.



The following is a complete list of available graphs.

- **Clumpy Scatterplots**: Clumpy scatterplots are 2D plots with evident clusters. These clusters are regions of high point density separated from other regions of points. The clusters can have many different shapes and are not necessarily circular. All possible scatterplots based on pairs of features (variables) are examined for clumpiness. The displayed plots are ranked according to the RUNT statistic. Note that the test for clumpiness is described in Hartigan, J. A. and Mohanty, S. (1992), "The RUNT test for multimodality," Journal of Classification, 9, 63–70 ([7]). The algorithm implemented here is described in Wilkinson, L., Anand, A., and Grossman, R. (2005), "Graph-theoretic Scagnostics," in Proceedings of the IEEE Information Visualization 2005, pp. 157–164 ([11]).

- **Correlated Scatterplots**: Correlated scatterplots are 2D plots with large values of the squared Pearson correlation coefficient. All possible scatterplots based on pairs of features (variables) are examined for correlations. 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. When modeling with these variables, you may want to leave out variables that are perfectly correlated with others.

- **Unusual Scatterplots**: Unusual scatterplots are 2D plots with features not found in other 2D plots of the data. The algorithm implemented here is described in Wilkinson, L., Anand, A., and Grossman, R. (2005), "Graph-theoretic Scagnostics," in Proceedings of the IEEE Information Visualization 2005, pp. 157-164. Nine scagnostics ("Outlying", "Skewed", "Clumpy", "Sparse", "Striated", "Convex", "Skinny", "Stringy", "Correlated") are computed for all pairs of features. The scagnostics are then examined for outlying values of these scagnostics and the corresponding scatterplots are displayed.

- **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 ([5]). 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, 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 ([10]). 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.

- **Radar Plot**: A Radar Plot is a 2D 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 blue-ish and large values are red-ish. 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.

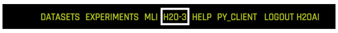
- **Missing Values Heatmap**: The missing values heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables and columns represent cases (instances). The data are coded into the values 0 (missing) and 1 (nonmissing). Missing values are colored red and nonmissing 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.

The images on this page are thumbnails. You can click on any of the graphs to view and download a full-scale image.

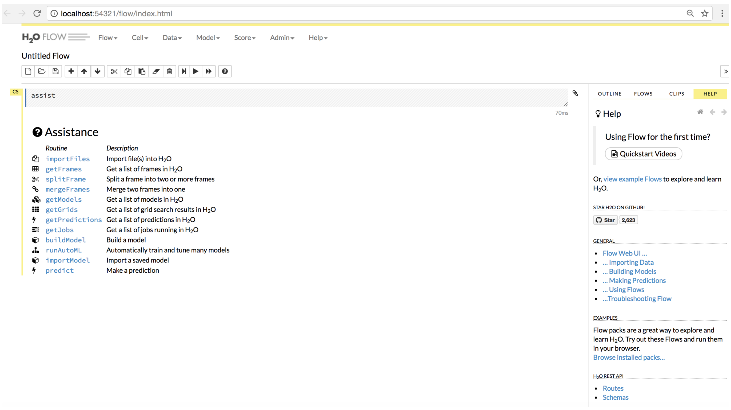
The correlation network graph is constructed from all pairwise squared correlations between variables (features). For continuous-continuous variable pairs, the statistic used is the squared Pearson correlation. For continuous-categorical variable pairs, the statistic is based on the squared intraclass 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.

Launching H2O Flow

If you opened port 54321 when starting Driverless AI, then you can launch H2O Flow from within Driverless AI. Click the **H2O-3** link in the top menu.



This launches Flow on port 54321.



```
assist
```

Assistance

Routine	Description
importFiles	Import file(s) into H_2O
getFrames	Get a list of frames in H_2O
splitFrame	Split a frame into two or more frames
mergeFrames	Merge two frames into one
getModels	Get a list of models in H_2O
getGrids	Get a list of grid search results in H_2O
getPredictions	Get a list of predictions in H_2O
getJobs	Get a list of jobs running in H_2O
buildModel	Build a model
runAutoML	Automatically train and tune many models
importModel	Import a saved model
predict	Make a prediction

Help

Using Flow for the first time?

Quickstart Videos

Or, view example Flows to explore and learn H_2O.

STAR H2O ON GITHUB!

Star 2,623

GENERAL

- Flow Web UI ...
- ... Importing Data
- ... Building Models
- ... Making Predictions
- ... Using Flows
- ...Troubleshooting Flow

EXAMPLES

Flow packs are a great way to explore and learn H_2O. Try out these Flows and run them in your browser.
Browse installed packs...

H_2O REST API

- Routes
- Schemas

About Driverless AI Transformations

Transformations in Driverless AI are applied to columns in the data. The transformers create the engineered features. Driverless AI provides the following transformers:

- **Filter Transformer**: The Filter Transformer counts each numeric value in the dataset.
- **Frequent Transformer**: The Frequent Transformer counts each categorical value in the dataset. This count can be either the raw count or the normalized count.
- **Bulk Interactions Transformer**: The Bulk Interactions Transformer will add, divide, multiply, and subtract two columns in the data.
- **Truncated SVD Numeric Transformer**: Truncated SVD trains on a selected numeric of columns in the data. The components of the truncated SVD will be new features.
- **Cross Validation Target Encoding**: Cross validation target encoding is done on a categorical column.

- **Cross Validation Categorical to Numeric Encoding**: This transformer converts a categorical column to a numeric column. Cross validation target encoding is done on the categorical column.
- **Dates Transformer**: The Dates Transformer retrieves any date values, including:
  - Year
  - Quarter
  - Month
  - Day
  - Day of year
  - Week
  - Week day
  - Hour
  - Minute
  - Second
- **Date Polar Transformer**: The Date Polar Transformer expands the date using polar coordinates. The Date Transformer will only expand the date into different units, for example month. This does not capture the similarity between the months December and January (12 and 1) or the hours 23 and 0. The polar coordinates capture the similarities between these cases by representing the unit of the date as a point in a cycle. For example, the polar coordinates of: get minute in hour, would be the minute hand position on a clock.
- **Text Transformer**: The Text Transform transforms a text column using TFIDF (term frequency-inverse document frequency) or count (count of the word). This may be followed by dimensionality reduction using truncated SVD.
- **Numeric to Categorical Target Encoding Transformer**: This transformer converts a numeric column to categorical by binning. Cross validation target encoding is done on the binned column.
- **Cluster Target Encoding Transformer**: Selected columns in the data are clustered, and target encoding is done on the cluster ID.
- **Cluster Distance Transformer**: Selected columns in the data are clustered, and the distance to a chosen cluster center is calculated.

- **Weight of Evidence**: Creates likelihood type of features using the Weights Of Evidence (WOE) transformation method. The weight of evidence tells the predictive power of an independent variable in relation to the dependent variable, for example, the measurement of good customers in relations to bad customers.

$$\text{WOE} = \ln\left(\frac{\text{Distribution of Goods}}{\text{Distribution of Bads}}\right)$$

  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/.

- **Numeric To Categorical Weight of Evidence Transformer**: This transformer converts a numeric column to categorical by binning and then creates the likelihood type of features using the WOE transformation method.

Example Transformations

In this section, we will describe the transformations using the example of predicting house prices on the example dataset.

Date Built	Square Footage	Num Beds	Num Baths	State	Price
01/01/1920	1700	3	2	NY	$700K

Frequent Transformer

- the count of each categorical value in the dataset
- the count can be either the raw count or the normalized count

Date Built	Square Footage	Num Beds	Num Baths	State	Price	Freq_State
01/01/1920	1700	3	2	NY	700,000	4,500

There are 4,500 properties in this dataset with state = NY.

Bulk Interactions Transformer

- add, divide, multiply, and subtract two columns in the data

Date Built	Square Footage	Num Beds	Num Baths	State	Price	Interaction_NumBeds#subtract#NumBaths
01/01/1920	1700	3	2	NY	700,000	1

There is one more bedroom than there are number of bathrooms for this property.

Truncated SVD Numeric Transformer

- truncated SVD trained on selected numeric columns of the data
- the components of the truncated SVD will be new features

Date Built	Square Footage	Num Beds	Num Baths	State	Price	TruncSVD_Price_NumBeds_NumBaths_1
01/01/1920	1700	3	2	NY	700,000	0.632

The first component of the truncated SVD of the columns Price, Number of Beds, Number of Baths.

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

Date Built	Square Footage	Num Beds	Num Baths	State	Price	DateBuilt_Month
01/01/1920	1700	3	2	NY	700,000	1

The home was built in the month January.

Date Polar Transformer

- get hour in day, get minute in hour, get day in month, get day in year, get quarter in year, get month in year, get week in year, get week day in week

Date Built	Square Footage	Num Beds	Num Baths	State	Price	Date-Built_MonthInYear_x	Date-Built_MonthInYear_y
01/01/1920	1700	3	2	NY	700,000	0.5	1

The polar coordinates of the month January in year is (0.5, 1). This allows the model to catch the similarities between January and December. This information was not captured in the simple Date Transformer.

Text Transformer

- transform text column using methods: TFIDF or count (count of the word)
- this may be followed by dimensionality reduction using truncated SVD

Categorical Target Encoding Transformer

- cross validation target encoding done on a categorical column

Date Built	Square Footage	Num Beds	Num Baths	State	Price	CV_TE_State
01/01/1920	1700	3	2	NY	700,000	550,000

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.

Numeric to Categorical Target Encoding Transformer

- numeric column converted to categorical by binning
- cross validation target encoding done on the binned numeric column

Date Built	Square Footage	Num Beds	Num Baths	State	Price	CV_TE_SquareFootage
01/01/1920	1700	3	2	NY	700,000	345,000

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.

Cluster Target Encoding Transformer

- selected columns in the data are clustered
- target encoding is done on the cluster ID

Date Built	Square Footage	Num Beds	Num Baths	State	Price	ClusterTE_4_NumBeds_NumBaths_SquareFootage
01/01/1920	1700	3	2	NY	700,000	450,000

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.

Cluster Distance Transformer

- selected columns in the data are clustered
- the distance to a chosen cluster center is calculated

Date Built	Square Footage	Num Beds	Num Baths	State	Price	ClusterDist_4_NumBeds_NumBaths_SquareFootage_1
01/01/1920	1700	3	2	NY	700,000	0.83

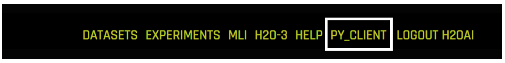
The columns: `Num Beds`, `Num Baths`, `Square Footage` have been segmented into 4 clusters. The difference from this record to Cluster 1 is 0.83.

Using the Driverless AI Python Client

This section describes how to run Driverless AI using the Python client.

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 **PY_CLIENT** link in the top menu of the UI.



Running an Experiment using the Python Client

1. After the h2oai_client wheel is installed, import the required modules and log in.

```
import h2oai_client
import numpy as np
import pandas as pd
import requests
import math
from h2oai_client import Client, ModelParameters, InterpretParameters

address = 'http://ip_where_driverless_is_running:12345'
username = 'username'
password = 'password'
h2oai = Client(address = address, username = username, password =
     password)
# Be sure to use the same credentials that you use when signing in
# through the GUI
```

2. Upload training, testing, and validation datasets from the Driverless AI **/data** folder. The validation and testing dataset are optional. This example shows how to add a validation dataset, but the experiment will only specify training and testing datasets.

```
train_path = '/data/CreditCard/CreditCard-train.csv'
test_path = '/data/CreditCard/CreditCard-test.csv'
valid_path = '/data/CreditCard/CreditCard-valid.csv'

train = h2oai.create_dataset_sync(train_path)
test = h2oai.create_dataset_sync(test_path)
valid = h2oai.create_dataset_sync(valid_path)
```

3. Set experiment parameters for the 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. This is set to "Disabled" if a validation set is used.
   - Time Column: The column that provides a time order, if applicable. If "AUTO", Driverless AI will auto-detect a potential time order. If "OFF", auto-detection is disabled. This is set to "Disabled" if a validation set is used.

For this example, we will be predicting **default payment next month**. We can set the parameters by hand or let Driverless AI infer the parameters and override any we disagree with. In this case, we will let Driverless AI suggest the best parameters for our experiment.

```
# let Driverless suggest parameters for experiment
target="default payment next month"
params = h2oai.get_experiment_tuning_suggestion(dataset_key = train.key
     , target_col = target)

params.dump()
{'accuracy': 7,
 'cols_to_drop': [],
 'dataset_key': 'corulege',
 'enable_gpus': True,
 'fold_col': '',
 'interpretability': 7,
 'is_classification': True,
 'scorer': '',
 'seed': False,
 'target_col': 'default payment next month',
 'testset_key': '',
 'time': 3,
 'time_col': '',
 'validset_key': '',
 'weight_col': ''}
```

Driverless AI has found that the best parameters are to set `accuracy = 7`, `time = 3`, and `interpretability = 7`. We will add our test data to the parameters, set the scorer to AUC, and add a seed to make the experiment reproducible.

4. Specify the experiment settings. Refer to the Experiment Settings for more information about these settings.

```
params.testset_key = test.key
params.scorer = "AUC"
params.seed = 1234
```

5. Launch the experiment to run feature engineering and final model training. In addition to the settings previously defined, be sure to also specify the imported training dataset. Adding a test dataset is optional, but a validation dataset is not.

```
experiment = h2oai.start_experiment_sync(params)
```

6. Examine 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.

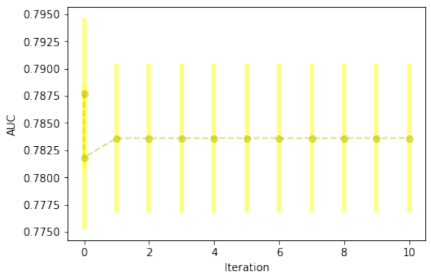
```
print("Final Model Score on Validation Data: " + str(round(experiment.
     train_score, 3)))
```

```
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: 0.802
```

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).

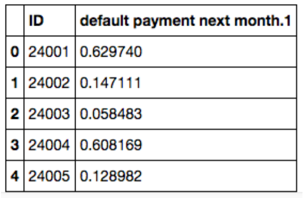
This information is saved in the experiment object.

```
import matplotlib.pyplot as plt

iterations = list(map(lambda iteration: iteration.iteration, experiment
     .iteration_data))
scores_mean = list(map(lambda iteration: iteration.score_mean,
     experiment.iteration_data))
scores_sd = list(map(lambda iteration: iteration.score_sd, experiment.
     iteration_data))

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(scorer_str)
plt.show();
```



7. 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.

```
h2oai.download(src_path=experiment.test_predictions_path, dest_dir=".")
'./test_preds.csv'

test_preds = pd.read_csv("./test_preds.csv")
test_preds.head()
```



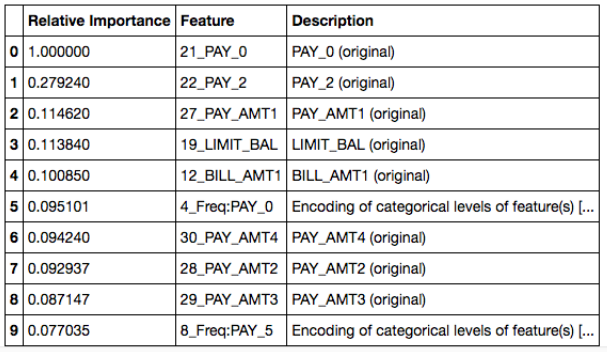
	ID	default payment next month.1
0	24001	0.629740
1	24002	0.147111
2	24003	0.058483
3	24004	0.608169
4	24005	0.128982

8. We can also download and examine the features and their importance for the final model.

```
# Download Summary
import subprocess
summary_path = h2oai.download(src_path=experiment.summary_path,
     dest_dir=".")
dir_path = "./h2oai_experiment_summary_" + experiment.key
subprocess.call(['unzip', '-o', summary_path, '-d', dir_path], shell=
     False)
```

The table below shows the feature name, its relative importance, and a description. Some features will be engineered by Driverless AI and some can be the original feature.

```
# View Features
features = pd.read_table(dir_path + "/features.txt", sep=',',
     skipinitialspace=True)
features.head(n = 10)
```



	Relative Importance	Feature	Description
0	1.000000	21_PAY_0	PAY_0 (original)
1	0.279240	22_PAY_2	PAY_2 (original)
2	0.114620	27_PAY_AMT1	PAY_AMT1 (original)
3	0.113840	19_LIMIT_BAL	LIMIT_BAL (original)
4	0.100850	12_BILL_AMT1	BILL_AMT1 (original)
5	0.095101	4_Freq:PAY_0	Encoding of categorical levels of feature(s) [...
6	0.094240	30_PAY_AMT4	PAY_AMT4 (original)
7	0.092937	28_PAY_AMT2	PAY_AMT2 (original)
8	0.087147	29_PAY_AMT3	PAY_AMT3 (original)
9	0.077035	8_Freq:PAY_5	Encoding of categorical levels of feature(s) [...

Access an Experiment Object that was Run through the Web UI

It is also possible to use the Python API to examine an experiment that was started through the Web UI using the experiment key.

You can get a pointer to the experiment by referencing the experiment key in the Web UI.

```
# Get a list of experiments
experiment_list = list(map(lambda x: x.key, h2oai.list_models(offset=0, limit
     =100)))
experiment_list
['fapudage', 'cunegacu', 'hamasida']

# Get pointer to experiment
experiment = h2oai.get_model_job(experiment_list[0]).entity
```

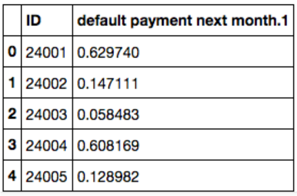
Score on New Data

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`.

The following shows the predicted probability of default for each record in the test.

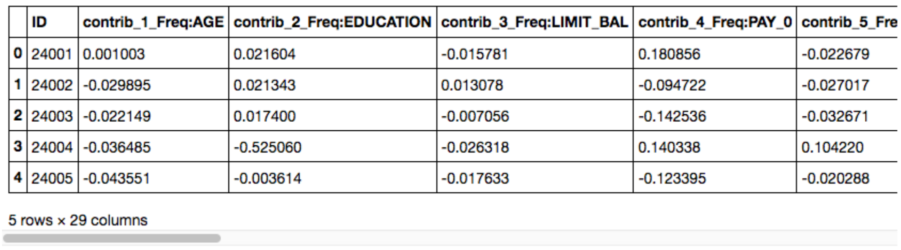
```
prediction = h2oai.make_prediction_sync(experiment.key, test_path,
     output_margin = False, pred_contribs = False)
pred_path = h2oai.download(prediction.predictions_csv_path, '.')
pred_table = pd.read_csv(pred_path)
pred_table.head()
```



	ID	default payment next month.1
0	24001	0.629740
1	24002	0.147111
2	24003	0.058483
3	24004	0.608169
4	24005	0.128982

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.

```
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()
```

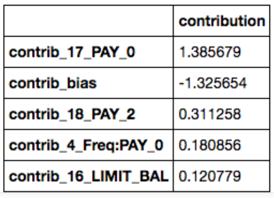


	ID	contrib_1_Freq:AGE	contrib_2_Freq:EDUCATION	contrib_3_Freq:LIMIT_BAL	contrib_4_Freq:PAY_0	contrib_5_Fre
0	24001	0.001003	0.021604	-0.015781	0.180856	-0.022679
1	24002	-0.029895	0.021343	0.013078	-0.094722	-0.027017
2	24003	-0.022149	0.017400	-0.007056	-0.142536	-0.032671
3	24004	-0.036485	-0.525060	-0.026318	0.140338	0.104220
4	24005	-0.043551	-0.003614	-0.017633	-0.123395	-0.020288

5 rows × 29 columns

We will examine the contributions for our first record more closely.

```
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()
```



	contribution
contrib_17_PAY_0	1.385679
contrib_bias	-1.325654
contrib_18_PAY_2	0.311258
contrib_4_Freq:PAY_0	0.180856
contrib_16_LIMIT_BAL	0.120779

This customer's `PAY_0` value had the greatest impact on their prediction. Because the contribution is positive, we know that it increases the prediction that they will default.

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. You can run model interpretation on raw data and on external model predictions.

Run Model Interpretation on 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. If you set `use_raw_features` to False, the model will be interpreted using the features used in the final model (raw and engineered).

Note that you can also specify the number of cross-validation folds to use in K-LIME. This defaults to 0.

```
mli_experiment = h2oai.run_interpretation_sync(
InterpretParameters(dai_model_key=experiment.key,
                      dataset_key=train.key,
                      target_col=target,
                      prediction_col = "",
                      use_raw_features=True, # show interpretation based on the
                            original columns
                      klime_cluster_col='',
                      nfolds = 0 # number of folds used for k-lime
                     ))
```

You can also see the list of interpretations using the Python client.

```
# Get list of interpretations
mli_list = list(map(lambda x: x.key, h2oai.list_interpretations(offset=0,
     limit=100)))
mli_list

['wekature', 'bawamoci']
```

Run Model Interpretation on External Model Predictions

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. 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.

```
# Dataset must be located where python client is running
train_pd = pd.read_csv(train_path)

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])
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)

predictions = gbm_model.predict_proba(train_pd[predictors])
predictions[0:5]
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 `2o_ai.run_interpretation_sync` to create the interpretation screen.

```
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(train_gbm_path)

mli_external = h2oai.run_interpretation_sync(
    InterpretParameters(dai_model_key="",
                        dataset_key=train_gbm_pred.strip("\""),
                        target_col=target,
                        prediction_col = "p1",
                        use_raw_features=True, # not relevant since we are
                             interpreting our external model
                        klime_cluster_col='',
                        nfolds = 0 # number of folds used for k-lime
                       ))
```

FAQ

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.

How can I upgrade to a newer version of Driverless AI?

H2O provides the following set of commands that can you can run on the machine that's running Driverless AI. Note that these commands only work on Linux environments. These commands are not available for Mac and Windows installations.

```
h2oai stop     (Stop all instances)
h2oai start    (Start an instance)
h2oai restart  (Restart instance)
h2oai clean    (Removes old containers)
h2oai purge    (Removes old containers and purges tmp and log)
```

```
h2oai upgrade  (Upgrades image to Developer latest *unstable*)
h2oai update   (Upgrades image to latest Release)
h2oai ssh      (Attaches to running docker)
h2oai log      (Tails the running server log)
h2oai jupyter  (Fetch the Jupyter URL with token)
```

To upgrade to the latest version, stop Driverless AI, then run the `h2oai update` command.

What kind of authentication is supported in Driverless AI?

Driverless AI supports basic, LDAP, and PAM authentication. This 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 the Driverless AI User Guide for installation and starting instructions.)

```
#Authentication
#authentication_method: available options are "none", "basic", "pam", "ldap"
authentication_method = "basic"

#Ldap Settings
ldap_server = ""
ldap_port = ""
ldap_dc = ""

# Configurations for a HDFS data source
# Path of hdfs coresite.xml
core_site_xml_path = ""
# Path of the dai principal key tab file
key_tab_path = ""

# HDFS connector
# Auth type can be Principal/keytab/keytabPrincipal
# Specify HDFS Auth Type, allowed options are described below
#   Principal : Authenticate with HDFS with a principal user
#   Keytab : Authenticate with a Key tab, preferably the keytab is created
     for the dai application
#   Impersonate with Keytab : Login with Impersonation using a dai keytab
hdfs_auth_type = "keytab"
# DAI recomends a user to be created in kerberos for the driverless ai
     application
# specificy the kerberos app principal user below
hdfs_app_principal_user = ""
# Specify the user id of the current user here as user@realm
hdfs_app_login_user = ""
#
hdfs_app_jvm_args = ""

#S3 Connector credentials
aws_access_key_id = ""
aws_secret_access_key = ""
```

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> below with your server name.

```
server {
    listen 80;
    return 301 https://$host$request_uri;
}

server {
    listen 443;

    # Specify your server name here
    server_name <server_name>;

    ssl_certificate           /etc/nginx/cert.crt;
    ssl_certificate_key       /etc/nginx/cert.key;
    ssl on;
    ssl_session_cache  builtin:1000  shared:SSL:10m;
    ssl_protocols  TLSv1 TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!aNULL:!eNULL:!EXPORT:!CAMELLIA:!DES:!MD5:!PSK:!RC4;
    ssl_prefer_server_ciphers on;

    access_log            /var/log/nginx/dai.access.log;

    location / {
      proxy_set_header        Host $host;
      proxy_set_header        X-Real-IP $remote_addr;
      proxy_set_header        X-Forwarded-For $proxy_add_x_forwarded_for;
      proxy_set_header        X-Forwarded-Proto $scheme;

      # Fix the ?It appears that your reverse proxy set up is broken" error.
      proxy_pass          http://localhost:12345;
      proxy_read_timeout  90;

      # Specify your server name for the redirect
      proxy_redirect      http://localhost:12345 https://<server_name>;
    }
}
```

More information about SSL for Nginx in Ubuntu 16.04 can be found here: https://www.digitalocean.com/community/tutorials/how-to-create-a-self-signed-ssl-certificate-for-nginx-in-ubuntu-16-04.

Is there a file size limit for datasets?

The file size for datasets is limited by GPU memory, but we continue to make optimizations for getting more data into an experiment.

How does Driverless AI detect the ID column?

The ID column logic is that the column is named 'id', 'Id', 'ID' or 'iD' exactly. (It does not check the number of unique values.) For now, if you want to ensure that your ID column is downloaded with the predictions, then you would want to name it one of those names.

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 clustering, we impute missing values; for frequency encoding, we count the number of rows that have a certain missing feature.

If I drop several columns from the Train dataset, will Driveless AI understand that it needs to drop the same columns from the Test dataset?

If you drop columns from the dataset, Driverless AI will do the same on the test dataset.

Which algorithms are used in Driverless AI?

Currently we only use XGBoost GPU for building models and testing the engineered features. Additional GPU algorithms will be added at a later date.

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 close to iid
  - Internal holdouts are used for parameter tuning
  - 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 drift in the data
  - No training data wasted during training since training data not used for parameter tuning
  - Entire validation set used for parameter tuning

  - No CV possible, since we explicitly do not want to overfit on the training data

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, ensuring that the model build will stop when it ceases to improve. And additional steps to prevent overfitting include checking for IID and avoiding leakage during feature engineering.

A blog post describing Driverless AI overfitting protection in greater detail is currently in development.

What can I do if my training and validation data points are not identical?

If you feel that training and validation data points are not identical, you can optionally provide an observation weights column (such as exponential weighting in time, different weights for valid vs train, etc.). All of our algorithms and metrics in Driverless AI support observation weights.

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, since we do up to 5-fold CV.)

Where can I get details of the various transformations performed in an experiment?

Inside the **/tmp** folder, you will see a folder with your experiment ID, and within that folder is a *logs_<experiment>.zip file. This zip file includes summary information, log information, and a gene_summary.txt file with details of the transformations used in the experiment.

How can I download the predictions onto the machine where Driverless AI is running?

When you select **Score on Another Dataset** the predictions will be automatically downloaded to the machine where Driverless AI is running. It will be saved in the following locations:

- Training Data Predictions: *tmp/experiment_name/train_preds.csv*
- Testing Data Predictions: *tmp/experiment_name/test_preds.csv*
- New Data Predictions: *tmp/experiment_name/automatically_generated_name*. Note that the automatically generated name will match the name of the file downloaded to your local computer.

References

1. L. Breiman. Statistical modeling: The two cultures (with comments and a rejoinder by the author). *Statistical Science*, 16(3), 2001. URL https://projecteuclid.org/euclid.ss/1009213726

2. M. W. Craven and J. W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, 1996. URL http://papers.nips.cc/paper/1152-extracting-tree-structured-representations-of-trained-networks.pdf

3. J. Friedman, T. Hastie, and R. Tibshirani. **The Elements of Statistical Learning**. Springer, New York, 2001. URL https://web.stanford.edu/~hastie/ElemStatLearn/printings/ESLII_print12.pdf

4. A. Goldstein, A. Kapelner, J. Bleich, and E. Pitkin. Peeking inside the black box: Visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, 24(1), 2015

5. R. A. Groeneveld and G. Meeden. **Measuring Skewness and Kurtosis**. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 33 (4):391–399, December 1984

6. P. Hall, W. Phan, and S. S. Ambati. Ideas on interpreting machine learning. *O'Reilly Ideas*, 2017. URL https://www.oreilly.com/ideas/ideas-on-interpreting-machine-learning

7. J. Hartigan and S. Mohanty. **The RUNT test for Multimodality**. *Journal of Classification*, 9(1):63–70, January 1992

8. J. Lei, M. G'Sell, A. Rinaldo, R. J. Tibshirani, and L. Wasserman. Distribution-free predictive inference for regression. *Journal of the American Statistical Association just-accepted*, 2017. URL http://www.stat.cmu.edu/~ryantibs/papers/conformal.pdf

9. M. T. Ribeiro, S. Singh, and C. Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016. URL http://www.kdd.org/kdd2016/papers/files/rfp0573-ribeiroA.pdf

10. L. Wilkinson. **Dot Plots**. *The American Statistician*, 53(3):276–281, 1999

11. L. Wilkinson, A. Anand, and R. Grossman. *"Graph-theoretic Scagnostics," in Proceedings of the IEEE Information Visualization*. INFOVIS '05. IEEE Computer Society, Washington, DC, USA, 2005

Authors

Patrick Hall

Patrick Hall is senior director for data science products at H2O.ai where he focuses mainly on model interpretability. Patrick is also currently an adjunct professor in the Department of Decision Sciences at George Washington University, where he teaches graduate classes in data mining and machine learning. Prior to joining H2O.ai, Patrick held global customer facing roles and research and development roles at SAS Institute.

Follow him on Twitter: @jpatrickhall

Megan Kurka

Megan is a customer data scientist at H2O.ai. Prior to working at H2O.ai, she worked as a data scientist building products driven by machine learning for B2B customers. Megan has experience working with customers across multiple industries, identifying common problems, and designing robust and automated solutions.

Angela Bartz

Angela is the doc whisperer at H2O.ai. With extensive experience in technical communication, she brings our products to life by documenting the features and functionality of the entire suite of H2O products. Having worked for companies both large and small, she is an expert at understanding her audience and translating complex ideas into consumable documents. Angela has a BA degree in English from the University of Detroit Mercy.