Using H2O Driverless AI

Patrick Hall, Megan Kurka, & Angela Bartz

http://docs.h2o.ai

November 2017: Version 1.0.5

Interpreting Machine Learning with H2O Driverless AI
by Patrick Hall, Megan Kurka & Angela Bartz

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

November 2017: Version 1.0.5

Printed in the United States of America.

Contents

1 Overview

Driverless AI seeks to build the fastest artificial intelligence (AI) platform on graphical processing units (GPUs). Many other machine learning algorithms can benefit from the efficient, fine-grained parallelism and high throughput of GPUs, which allow you to complete training and inference much faster than with CPUs.

In addition to supporting GPUs, Driverless AI also makes use of machine learning interpretability (MLI). Often times, especially in regulated industries, model transparency and explanation become just as important as predictive performance. Driverless AI utilizes MLI to include easy-to-follow visualizations, interpretations, and explanations of models.

This document describes how to install and use H2O Driverless AI and is based on a pre-release version. 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/.

1.1 Citation

To cite this booklet, use the following: Hall, P., Kurka, M., and Bartz, A. (Sept 2017). *Using H2O Driverless AI*. http://docs.h2o.ai

2 Installing Driverless AI

Installation 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). You should have lots of system CPU memory (64 GB or more) and free disk space (at least 10 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).

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

2.1 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 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				Install on Google Compute

Server

Operating System	GPUs?	Min Mem	Refer to Section
NVIDIA DGX-1	Yes	128 GB	Install on NVIDIA DGX
Ubuntu 16.0.4	Yes	64 GB	Install on Ubuntu with GPUs
Ubuntu	No	64 GB	Install on Ubuntu
RHEL	No	64 GB	Install on RHEL
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 DGX
Mac OS X	No	16 GB	Experimentation	Install on Mac OS X
Windows 10 Windows Server 2016	No	16 GB	Experimentation	Install on Windows
Windows 7	Not supported			
Linux	See **Server** table			

3 Running an Experiment

1. After Driverless AI is installed and started, open a Chrome browser and navigate to <driverless-ai-host-machine>:12345.

   Note: Driverless AI is only supported on Google Chrome.

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:

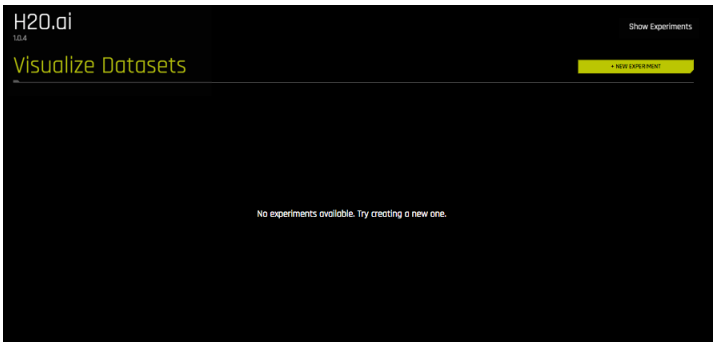
   ```
   Username: h2oai
   Password: h2oai
   ```

   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.

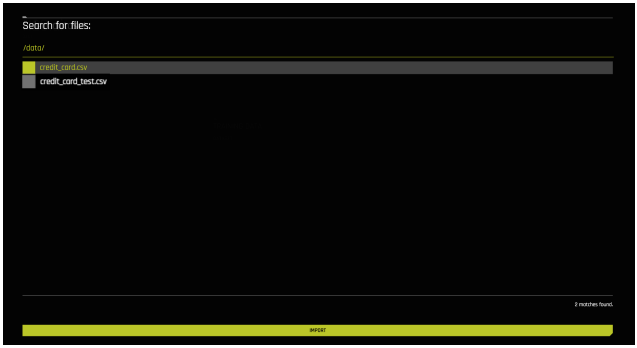
4. As with accepting the Evaluation Agreement, the first time you log in, you will be prompted to enter your License Key. Paste the License Key into the **License Key** entry field, and then click **Save** to continue. This license key will be saved in the host machine's **/license** folder that was created during installation.

   Note: Contact sales@h2o.ai for information on how to purchase a Driverless AI license.

5. The Home page appears, showing all experiments that have previously been run. Start a new experiment and/or add datasets by clicking the **New Experiment** button.



6. Click the **Select or import a dataset** button, then click the **Browse** button at the bottom of the screen. In the **Search for files** field, enter the location for the dataset. Note that Driverless AI autofills the browse line as type in the file location. When you locate the file, select it, then click **Import** at the bottom of the screen.

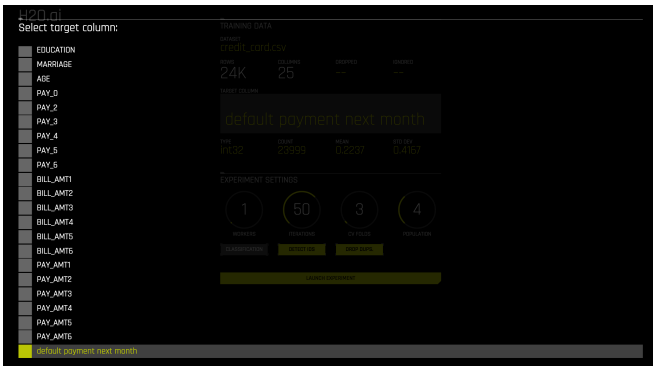


Note: To import additional datasets, click the **Show Experiments** link in the top-right corner of the UI, then click **New Experiment** again to browse and add another dataset.

7. Optionally specify whether to drop any columns (for example, an ID column).

8. Optionally specify a test dataset. Keep in mind that the test dataset must have the same number of columns as the training dataset.

9. Specify the target (response) column.



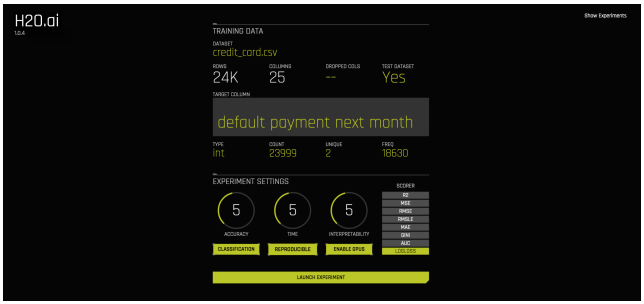
10. When the target column is selected, Driverless AI automatically provides the target column type and the number of rows. If this is a classification

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. Refer to About Experiment Settings for more information about these settings.

- Accuracy value (defaults to 5)
- Time setting (defaults to 5)
- Interpretability of the model (defaults to 5)
- Specify the scorer to use for this experiment. A scorer value is not selected by default.

Additional settings:

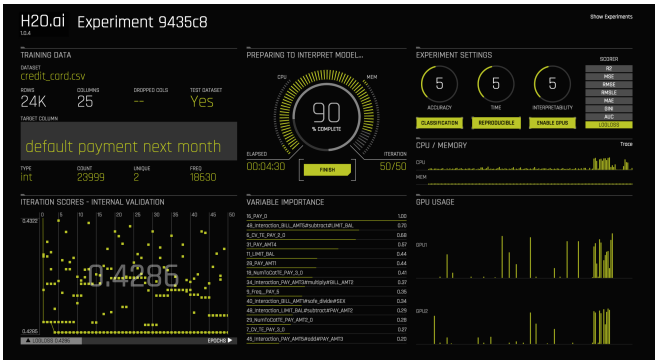
- If this is a classification problem, then click the **Classification** button.
- Click the **Reproducible** button to build this with a random seed.
- Specify whether to enable GPUs. (Note that this option is ignored on CPU-only systems.)



11. Click **Launch Experiment**. This starts the Driverless AI feature engineering process.

    As the experiment runs, a running status displays in the upper middle portion of the UI. In addition 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, and CPU/Memory and GPU Usage information.

    You can stop experiments that are currently running. Click the **Finish** button to end the experiment at its current spot and build a scoring package.



3.1 About Experiment Settings

This section describes the settings that are available when running an experiments.

Test Data

Test data is used to create test predictions only. This dataset is not used for model scoring.

Dropped Columns

Dropped columns are columns that you do not want to be used as predictors in the experiment.

Accuracy

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

Accuracy	Max Rows	Ensemble Level	Target Transformation	Tune Parameters	Num Individuals	CV Folds	Only First CV Model	Strategy
1	100K	0	False	False	Default	3	True	None
2	500K	0	False	False	Default	3	True	None
3	1M	1	False	False	Default	3	True	None
4	2.5M	1	False	False	Default	3	True	None
5	5M	1	True	False	Default	3	True	None
6	10M	2	True	True	Default	3	True	FS
7	20M	2	True	True	4	4	False	FS
8	20M	2	True	True	4	4	False	FS
9	20M	2	True	True	4	4	False	FS
10	None	2	True	True	8	4	False	FS

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
  - 0: single final model
  - 1: 3 3-fold final models ensembled together
  - 2: 5 5-fold final models ensembled together
- **Target Transformation:** Try target transformations and choose the transformation that has the best score
  - Possible transformations: identity, log, square, square root, inverse, Anscombe, logit, sigmoid
- **Tune Parameters:** Tune the parameters of the XGBoost model
  - Only `max_depth` tuned is tuned, and the range is 3 to 10
  - Max depth chosen by `penalized_score`, which is a combination of the model's accuracy 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.
  - Default is automatically determined. Typical values are 4 or 8.
- **CV Folds:** The number of cross validation folds done for each model
  - If the problem is a classification problem, then stratified folds are created.
- **Only First CV Model:** Equivalent to splitting data into a training and testing set
  - Example: Setting CV Folds to 3 and Only First CV Model = True means you are splitting the data into 66% training and 33% testing.
- **Strategy:** Feature selection strategy
  - None: No feature selection
  - FS: Feature selection permutations

Time

Time	Epochs
1	10
2	20
3	30
4	40
5	50
6	100
7	150
8	200
9	300
10	500

Interpretability

Interpretability	Strategy
<= 5	None
> 5	FS

4 Interpreting a Model

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

- Interpret this Model
- Score on Another Dataset
- Download (Holdout) Training Predictions (in csv format)
- Download Predictions (in csv format, available if a test dataset is used)
- Download Transformed Training Data (in csv format)
- Download Transformed Test Data (in csv format, available if a test dataset is used)
- Download Logs
- Download Scoring Package (a standalone Python scoring package for H2O Driverless AI)
- View Warnings (if any existed)

Click the **Interpret this Model** button to open the Model Interpretation page, which provides several visual explanations of the trained Driverless AI model and its results.

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.

4.1 K-LIME

4.1.1 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

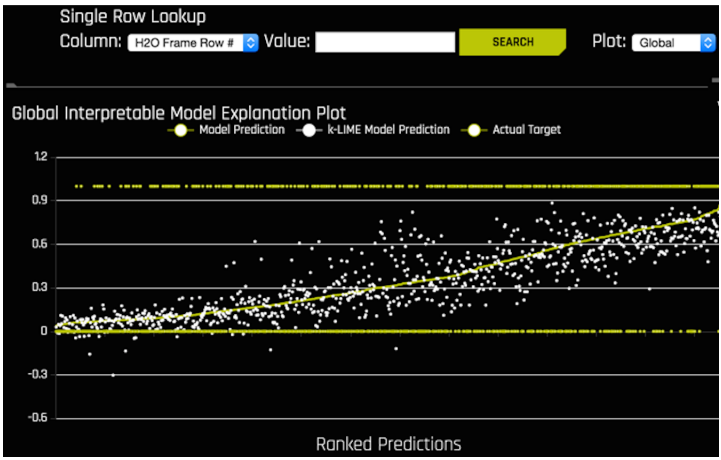
Each local contribution is positive and thus contributes positively to the Driverless AI model's prediction of 0.85 for *H2OAI_predicted_default*. By taking into consideration the value of each contribution, reason codes for the Driverless AI decision can be derived. *debt_to_income_ratio* and *credit_score* 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.

4.1.2 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.



4.2 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.

4.2.1 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.

4.2.2 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.

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

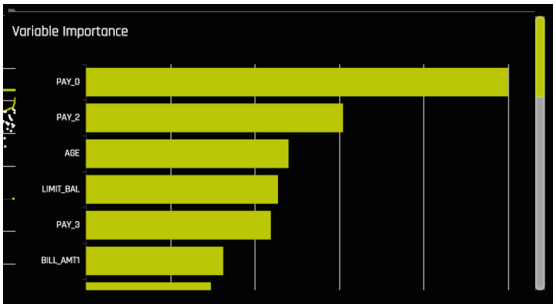
$$\text{Scaled}(\text{LOCO}_{credit\_score}) = \text{Abs}(\text{LOCO}_{credit\_score}/0.14) = 0.86$$

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

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.

4.2.3 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.



4.3 Decision Tree Surrogate Model

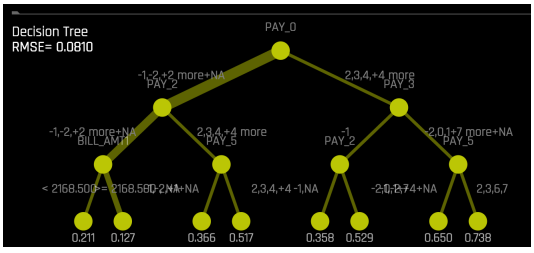
4.3.1 The Decision Tree Surrogate Model Technique

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.

4.3.2 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.



4.4 Partial Dependence and Individual Conditional Expectation (ICE)

4.4.1 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.

4.4.2 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(**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:

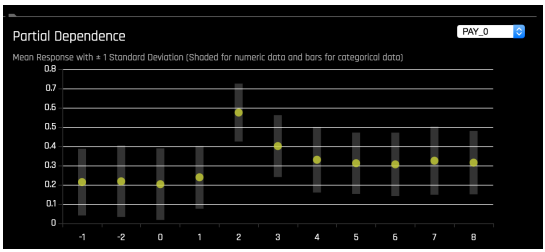
$$\text{ICE}_{credit\_score,500} = F(30, 500, 1000)$$
$$\text{ICE}_{credit\_score,530} = F(30, 530, 1000)$$
$$\text{ICE}_{credit\_score,560} = F(30, 560, 1000)$$
...
$$\text{ICE}_{credit\_score,800} = F(30, 800, 1000)$$

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.

4.4.3 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.



4.5 General Considerations

4.5.1 Machine Learning and Approximate Explanations

For years, common sense has deemed the complex, intricate formulas created by training machine learning algorithms to be uninterpretable. While great advances

have been made in recent years to make these often nonlinear, non-monotonic, and non-continuous machine-learned response functions more understandable (Hall et al, 2017 [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]).

4.5.2 The Multiplicity of Good Models in Machine Learning

It is well understood that for the same set of input variables and prediction targets, complex machine learning algorithms can produce multiple accurate models with very similar, but not exactly the same, internal architectures (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.

4.5.3 Expectations for Consistency Between Explanatory Techniques

- The decision tree surrogate is a global, nonlinear description of the Driverless AI model behavior. Variables that appear in the tree should have a direct relationship with variables that appear in the global 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.

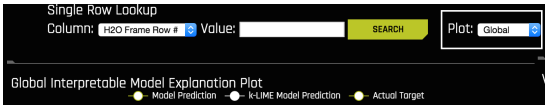
- ICE is a type of nonlinear sensitivity analysis which has a complex relationship to LOCO variable importance values. Comparing ICE to LOCO can only be done at the value of the selected variable that actually appears in the selected row of the training data. When comparing ICE to LOCO the total value 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.

5 Viewing Explanations

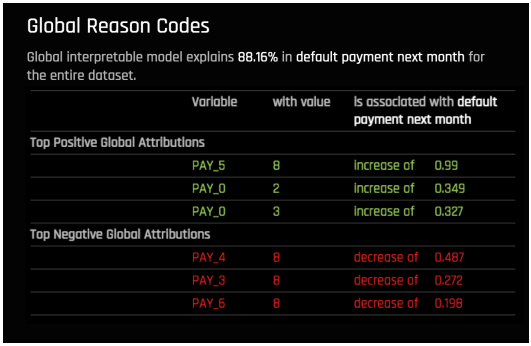
Driverless AI provides easy-to-read explanations for a completed model. You can view these by clicking the **Explanations** button in the upper-right corner of the Model Interpretation page. Note that this button is only available for completed experiments. Click **Close** when you are done to return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

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

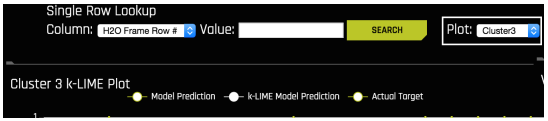
- **Global Reason Codes**: To view global reason codes, select the Global plot from the **Plot** dropdown.



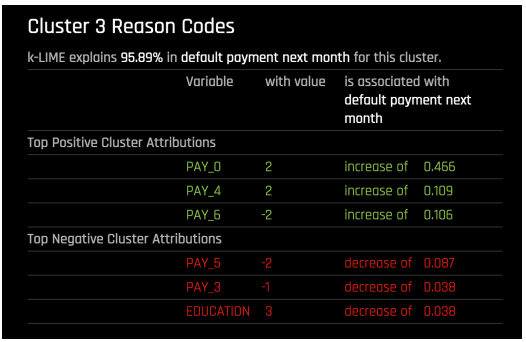
With Global selected, click the **Explanations** button in the upper-right corner.



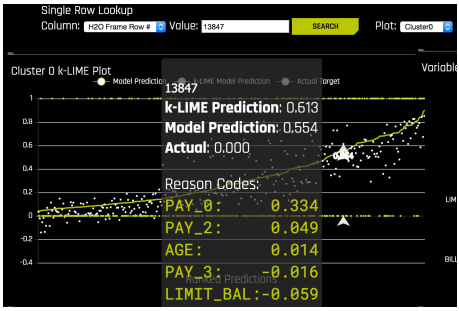
- **Cluster Reason Codes**: To view reason codes for a specific cluster, select a cluster from the **Plot** dropdown.



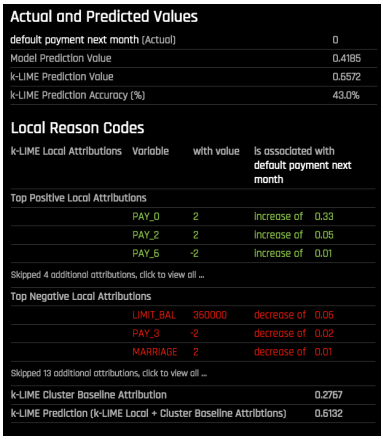
With a cluster selected, click the **Explanations** button in the upper-right corner.



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



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



6 The Scoring Package

As indicated earlier, a scoring package is available after a successfully completed experiment. This package includes a scoring module and a scoring service.

The scoring module is a Python module bundled into a standalone wheel file (named scoring_*.whl). All the prerequisites for the scoring module to work correctly are listed in the **requirements.txt** file.

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 modes:

- 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).

6.1 Prerequisites

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

- Linux x86_64 environment
- Python 3.6
- Virtual Environment
- Apache Thrift (to run the TCP scoring service)

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

Installing requirements on Ubuntu 16.10+

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

Installing requirements 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
```

Installing Thrift

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
```

6.2 Scoring Package Files

The **scoring-package** folder includes the following notable files:

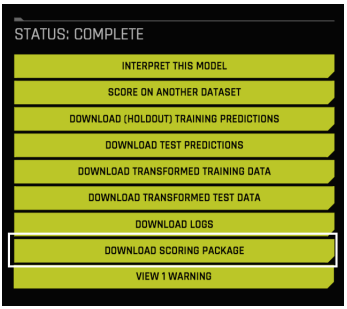
- **example.py**: An example Python script demonstrating how to import and score new records.
- **run_example.sh**: Runs example.py (also sets up a virtualenv with prerequisite libraries).
- **server.py**: A standalone TCP/HTTP server for hosting scoring services.
- **run_tcp_server.sh**: Runs TCP scoring 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).

6.3 Examples

This section provides examples showing how to run the scoring module and how to run the scoring service in TCP and HTTP mode.

Before running these examples, be sure that the scoring package is already downloaded and unzipped:

1. On the completed Experiment page, click on the **Download Scoring Package** button to download the **scorer.zip** file for this experiment onto your local machine.



2. Unzip the scoring package.

After the package is downloaded and unzipped, you will be able to run the scoring module and the scoring service.

6.3.1 Running the Scoring Module

Navigate to the **scoring-package** folder and run the following:

```
bash run_example.sh
```

The script creates a virtual environment within the **scoring-package** folder, installs prerequisites, and finally runs example.py, which uses the completed experiment.

```
StackedBaseModels transform
StackedBaseModels transform done
0.42403995659961626
StackedBaseModels transform
StackedBaseModels transform done
0.40546984142727327
StackedBaseModels transform
StackedBaseModels transform done
0.33347369233767193
StackedBaseModels transform
StackedBaseModels transform done
0.44764188594288296
StackedBaseModels transform
StackedBaseModels transform done
0.14722418040037155
StackedBaseModels transform
StackedBaseModels transform done
[ 0.42551939  0.42990264  0.34025302  0.46884989  0.15453387]
```

6.3.2 Running the Scoring Service - TCP Mode

TCP mode allows you to use the scoring service from any language supported by Thrift, including 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, generate the Thrift bindings once, and then run the server. 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.

```
# See the run_tcp_server.sh file for a complete example.
$ thrift --gen py scoring.thrift
$ python server.py --mode=tcp --port=9090
```

Call the scoring service by generating 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 the run_tcp_client.sh and example_client.py files for a complete example.
$ thrift --gen py scoring.thrift
   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()
```

Note that you can reproduce the exact same results from other languages. For example, to run the scoring service in Java, use:

```
$ thrift --gen java scoring.thrift
```

6.3.3 Running the Scoring Service - HTTP Mode

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.

See http://www.jsonrpc.org/specification for JSON-RPC documentation.

To start the scoring service in HTTP mode:

```
# See run_http_server.sh for a complete example
$ python server.py --mode=http --port=9090
```

To invoke scoring methods, compose a JSON-RPC message and make a HTTP POST request to http://host:port/rpc as follows:

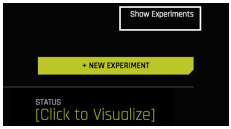
```
# See run_http_client.sh for a complete example
$ curl http://localhost:9090/rpc \
  --header "Content-Type: application/json" \
  --data @- <<EOF
  {
    "id": 1,
    "method": "score",
    "params": {
      "row": [ 7.486, 3.277, 4.755, 2.354 ]
    }
  }
  EOF
```

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:

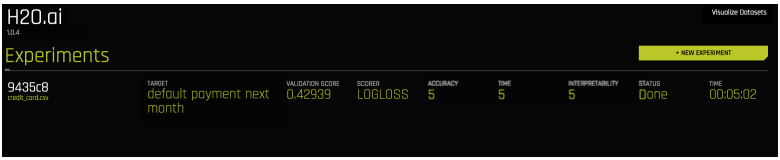
```
import requests
row = [7.486, 3.277, 4.755, 2.354]
req = dict(id=1, method='score', params=dict(row=row))
res = requests.post('http://localhost:9090/rpc', data=req)
print(res.json()['result'])
```

7 Viewing Experiments

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

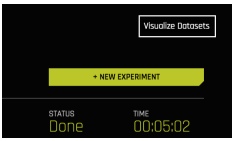


Click this link to open the Experiments page. From this page, you can select and view previous experiments, and you can begin a new experiment.

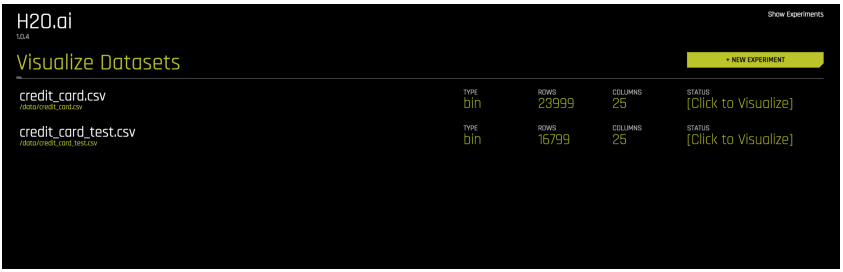


8 Visualizing Datasets

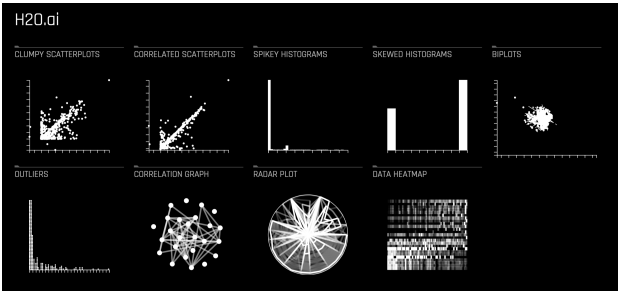
While viewing experiments, click the **Visualize Datasets** link in the upper-right corner.



The Datasets page shows a list of the datasets that you've imported.



Select a dataset to view the following graphical representations. Note that the list of graphs that displays can vary based on the information in your dataset.



- **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 red 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**: 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

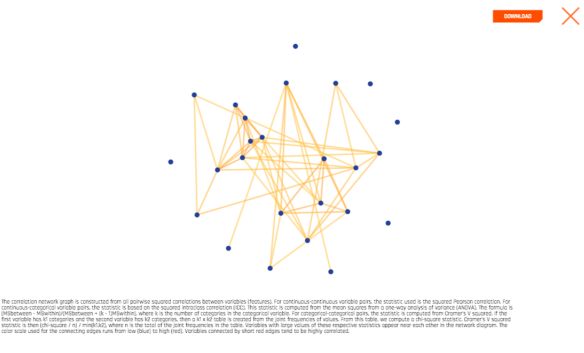
with these variables, you may want to leave out variables that are perfectly correlated with others.

- **Radar Plot**: A Radar Plot is a two-dimensional graph that is used for comparing multiple variables. Each variable has its own axis that starts from the center of the graph. The data are standardized on each variable between 0 and 1 so that values can be compared across variables. Each profile, which usually appears in the form of a star, connects the values on the axes for a single observation. Multivariate outliers are represented by red profiles. The Radar Plot is the polar version of the popular Parallel Coordinates plot. The polar layout enables us to represent more variables in a single plot.

- **Data Heatmap**: The heatmap graphic is constructed from the transposed data matrix. Rows of the heatmap represent variables, and columns represent cases (instances). The data are standardized before display so that small values are 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.



9 About Driverless AI Transformations

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

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

9.1 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.

9.2 Bulk Interactions Transformer

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

Date Built	Square Footage	Num Beds	Num Baths	State	Price	Interac-tion_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.

9.3 Truncated SVD Numeric Transformer

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

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.

9.4 Dates Transformer

- get year, get quarter, get month, get day, get day of year, get week, get week day, get hour, get minute, get second

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.

9.5 Date Polar Transformer

This transformer expands the date using polar coordinates. The Date Transformer (described above) 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.

- 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.

9.6 Text Transformer

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

9.7 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.

9.8 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.

9.9 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.

9.10 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	Cluster-Dist_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.

10 Using the Driverless AI Python Client

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

Notes:

- This is an early and experimental 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. Contact sales@h2o.ai for information on how to retrieve the h2oai_client wheel.

10.1 Running an Experiment using the Python Client

1. 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

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 and testing datasets from the Driverless AI **/data** folder.

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

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

3. Set the target (response) column and any ignored column or columns.

```
# set the parameters you want to pass to the UI
target = "default payment next month"
drop_cols = ['ID']
```

4. Specify the experiment settings. Refer to the :ref:'experimentsettings' for more information about these settings.

```
# Pre-set parameters to pass model
is_classification = True
enable_gpus = True
seed=True
scorer_str = 'auc'

# Pre-sent accuracy knobs
accuracy_value = 5
time_value = 5
interpretability = 1
```

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.

```
experiment = h2oai.start_experiment_sync(ModelParameters(
    dataset_key=train.key,
    testset_key=test.key,
    target_col=target,
    is_classification=is_classification,
    cols_to_drop=drop_cols,
    enable_gpus=enable_gpus,
    seed=seed,
    accuracy=accuracy_value,
    time= time_value,
    interpretability=interpretability,
    scorer=scorer_str
))
```

6. View the results for an iteration. Note that the Web UI shows a graph of the iteration scores. You can retrieve the scores of each iteration from the experiment object using the Python client. The example below retrieves the score for the last iteration:

```
score = experiment.iteration_data[-1].scores # gets the ScoresTable
score = score.score[-1]
print(score)

0.7875823819933607
```

7. View the final model score for the train 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 train and test data for this ensemble model.

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

Final Model Score on Train Data: 0.782
Final Model Score on Test Data: 0.803
```

8. Download the test 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()

default payment next month.1
0       0.514850
1       0.136738
2       0.062433
3       0.481917
4       0.126809
```

10.2 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.

```
experiment = h2oai.get_model_job("56507f").entity
```

10.3 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`.

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

default payment next month.1
0       0.514850
1       0.136738
2       0.062433
3       0.481917
4       0.126809
```

11 FAQ

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.

How can you 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.

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

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

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.

12 Have Questions?

If you have questions about using Driverless AI, post them on Stack Overflow using the **h2o** tag at http://stackoverflow.com/questions/tagged/h2o.

13 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 Amer-*

ican 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

14 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.