Machine learning using Stata via H2O

Zhao Xu

Principal Software Engineer
StataCorp LLC

Chinese Stata Conference
July 5, 2026

Outline

- Stata and H2O integration
- Ensemble decision trees
  - Gradient boosting machine and random forest
  - Model training and evaluation
  - Model selection and hyperparameter tuning
- Explainable machine learning
  - Model interpretability vs. explainability
  - Local and global explainable methods
  - Model specific vs. model agnostic explainable methods

Stata and H2O integration

Stata's integration of H2O machine learning provides a powerful, scalable, and user-friendly framework for applying modern machine learning techniques.

- **h2o** suite of commands for interacting with H2O cluster.
- **_h2oframe** suite of commands for working with H2O frames.
- **h2oml** suite of commands for end-to-end machine learning analysis.
  - Ensemble decision trees: Gradient boosting machine (GBM) and random forest for regression, binaray and multiclass classification
  - Estimation metric summaries
  - Model performance evaluation
  - Prediction
  - Machine learning explainability

What is H2O?

H2O is a scalable, distributed machine learning and predictive-analytics platform.

- Provides parallelized implementations of many widely used supervised and unsupervised algorithms.
- Runs as a cluster (a server running on the JVM) that performs the heavy computation – host either locally or on a remote server.
- Designed for multi-core / multi-node scaling.

H2O cluster

The H2O cluster is a process running within the Java Virtual Machine (JVM) environment. It hosts and manages all machine learning resources, including data and models.

H2O cluster	
h2o init	Create a new H2O cluster locally
h2o connect	Connect to an existing H2O cluster
h2o disconnect	Close the connection to an existing H2O cluster
h2o shutdown	Shut down the H2O cluster
h2o clear	Discard all resources within the H2O cluster

H2O frame

H2O frames live on the H2O cluster and do not exist in Stata's memory. Stata loads data into the H2O cluster, where it is stored in the form of an H2O frame.

H2O frame management

_h2oframe put	Load Stata's current dataset into an H2O frame
_h2oframe create	Create a new H2O frame
_h2oframe import	Import data files into an H2O frame
_h2oframe upload	Upload data files into an H2O frame
_h2oframe split	Split H2O frames
27 more...	

You can also leverage Stata's extensive data management commands for data exploratory, data cleaning and feature engineering.

Estimation commands

The **h2oml** suite of commands provides 6 estimation commands that implement gradient boosting and random forest regression, binary classification, and multiclass classification.

Supervised learning	
h2oml gbregress	Gradient boosting regression
h2oml gbbinclass	Gradient boosting binary classification
h2oml gbmulticlass	Gradient boosting multiclass classification
h2oml rfregress	Random forest regression
h2oml rfbinclass	Random forest binary classification
h2oml rfmulticlass	Random forest multiclass classification

- **validframe()** and **cv()** options to specify a validation frame and to perform cross-validation to control for overfitting.
- **tune()** and **stop()** options to tune hyperparameters and stop early for better model performance.

Estimation results and postestimation frame

Estimation results	
h2omlest store	Store the current (active) estimation results
h2omlest restore	Load results into the current (active) estimation results
h2omlest dir	Display a list of the stored estimates
h2omlest drop	Drop the specified stored estimation results
h2omlest clear	Drop all stored estimation results

Postestimation frame	
h2omlpostestframe	Specify frame for postestimation analysis

Tuning and estimation summaries

Tuning and estimation summaries

h2omlestat metrics	Display performance metrics
h2omlgof	Goodness of fit for machine learning methods
h2omlestat cvsummary	Display cross-validation summary
h2omlestat gridsummary	Display grid-search summary
h2omlexplore	Explore models after grid search
h2omlselect	Select model after grid search
h2omlgraph scorehistory	Produce score history plot

Performance evaluation

Performance after binary classification	
h2omlestat threshmetric	Display threshold-based metrics
h2omlestat confmatrix	Display confusion matrix
h2omlgraph prcurve	Produce precision–recall curve plot
h2omlgraph roc	Produce receiver operating characteristic (ROC) curve plot
Performance after multiclass classification	
h2omlestat aucmulticlass	Display AUC and AUCPR
h2omlestat hitratio	Display hit-ratio table
Diagnostic plots after regression	
h2omlgraph rvfplot	Produce residual-versus-fitted plot
h2omlgraph rvpplot	Produce residual-versus-predictor plot
Prediction	
h2omlpredict	Prediction of continuous responses, probabilities, and classes

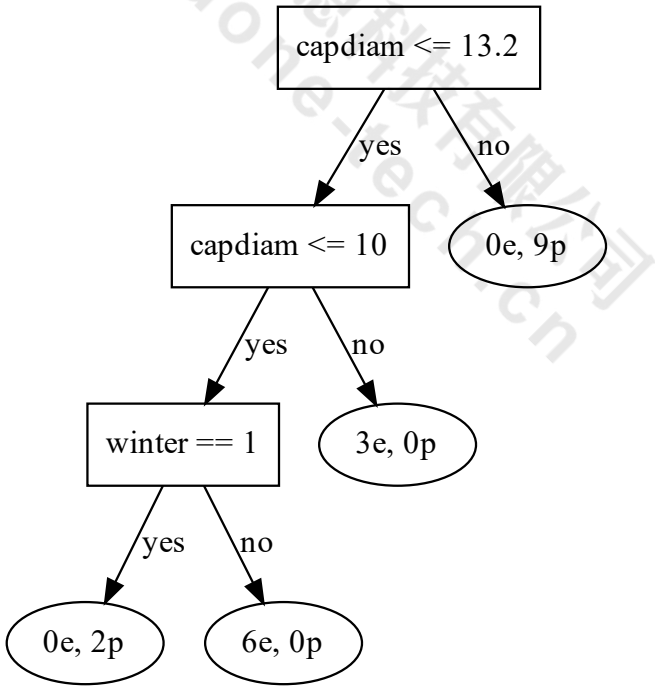
Machine learning explainability

Model explainability tools

h2omlgraph shapsummary	Produce SHAP beeswarm plot
h2omlgraph shapvalues	Produce SHAP values plot for individual observations
h2omlgraph pdp	Produce partial dependence plot
h2omlgraph ice	Produce individual conditional expectation plot
h2omlgraph varimp	Produce variable importance plot
h2omlgraph permimp	Produce permutation importance plot

Decision trees: the building block

A decision tree is a supervised ML model that makes predictions by recursively splitting the predictor space into non-overlapping regions, arriving at a prediction through a sequence of simple yes/no questions.



Decision trees: the building block

Pros:

- No need for data preprocessing - missing values/standardization/normalization
- Handles nonlinearity and interactions automatically.
- Handles mixed categorical and continuous predictors.
- Has high intrinsic interpretability.
- Can be used both for regression and classification.

Cons:

- **High variance/instability** – A small change in training data can produce a completely different tree.
- **Overfitting** – a tree grown deep enough can fit the training data almost perfectly while generalizing poorly to new data.
- **Selection bias** – trees can be biased toward dominant classes under class imbalance.

Ensemble methods

Let the data be drawn i.i.d. from an unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$. For a given training sample $S = \{(x_i, y_i)\}_{i=1}^{n}$, a learning algorithm produces a hypothesis $h : \mathcal{X} \to \mathcal{Y}$ from some hypothesis space $\mathcal{H}$. An ensemble constructs a set of base hypotheses $h_1, \ldots, h_T$ and combines them through a learned aggregation function F. For regression, this is commonly a weighted average,

$$H(x) = \sum_{t=1}^{T} w_t \, h_t(x), \qquad \sum_t w_t = 1, \;\; w_t \geq 0,$$

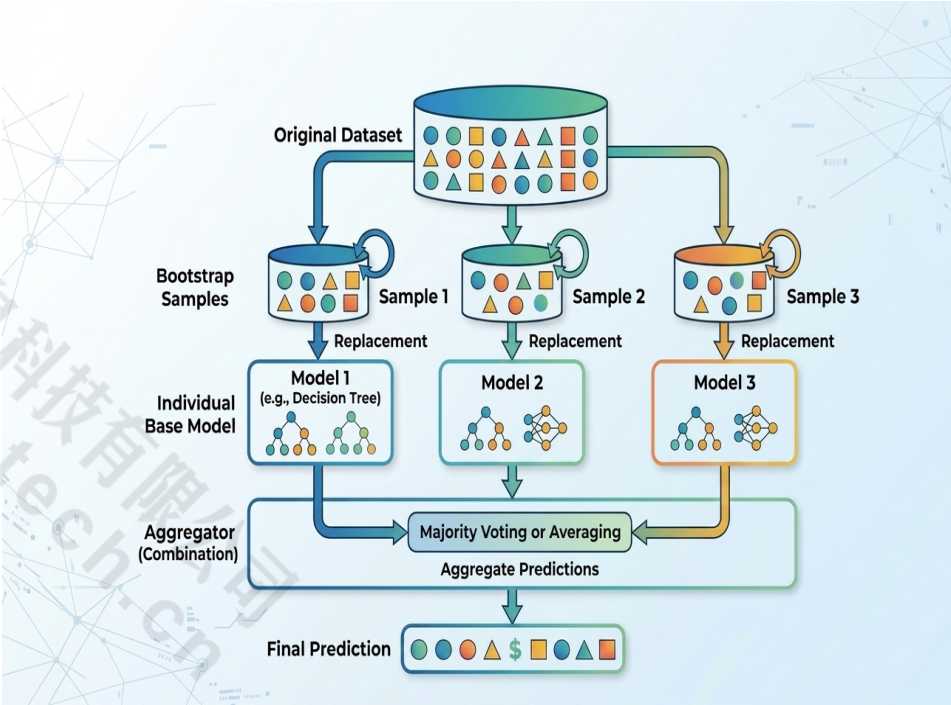
and for classification a weighted majority vote,

$$H(x) = \arg\max_{y \in \mathcal{Y}} \sum_{t=1}^{T} w_t 1[h_t(x) = y].$$

Bagging and **boosting** are two widely used ensemble techniques.

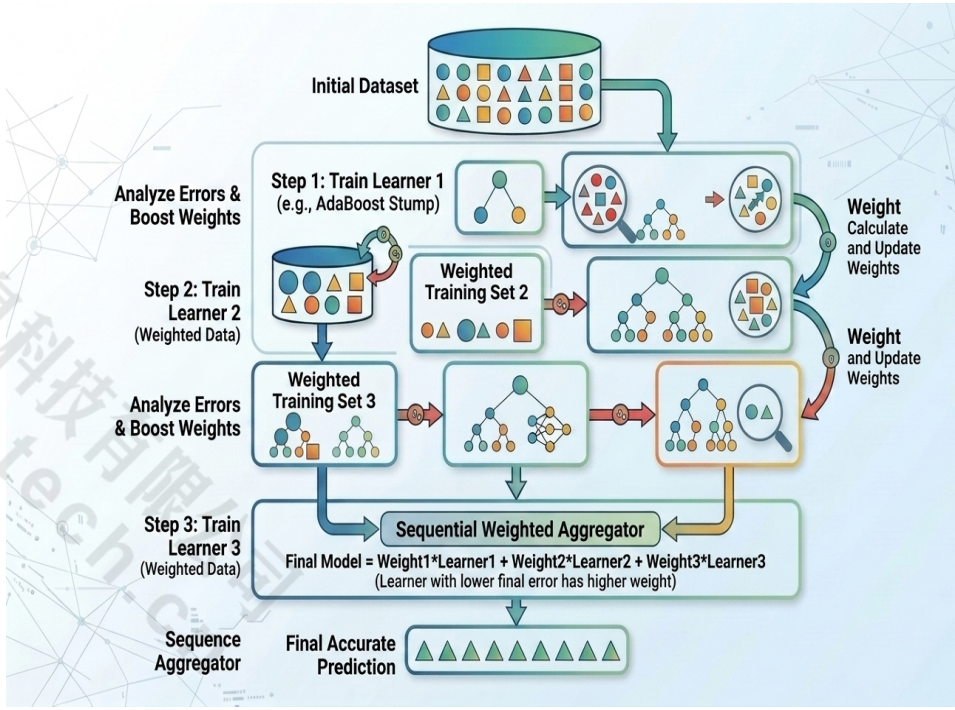
Bagging (Bootstrap Aggregating)

- Reduces model variance (overfitting) by training multiple models **independently**.
- Creates multiple subsets of the dataset (sampled with replacement), and trains an identical base model on each in **parallel**.
- Makes predictions by **averaging** the outputs of the trained base models (or majority vote for classification).
- Common algorithm: **Random Forest**.



Boosting

- Reduces model bias (a majority source of underfitting) by training models **sequentially**.
- Each new model focuses heavily on the observations that previous models predicted poorly.
- Makes decisions by combining the trained models using a weighted vote a weighted sum, building up the result through sequential correction.
- Common algorithms: **AdaBoost**, **Gradient Boosting (GBM)**, and **XGBoost**.



Running example: predict the churn status of customers

Consider data from a fictional company, Telco, that provides home phone and internet services in California. The data have been made available by IBM and contain **7,043** observations and **26** variables.

The binary response, **churn**, indicates whether a customer left within the last month or is still using Telco's services.

The predictors include:

- **demographic information**, such as gender and age;
- **account information**, such as payment period and duration of services;
- **service types**, such as whether a customer signed up for internet, phone, device protection, and so on.

Running example: our goal

- **Develop a robust, accurate prediction model to determine whether a customer will churn**. Reliable predictions improve a business's ability to intervene in time and retain at-risk customers.
- **Identify which predictors most influence churn**. This offers decision-makers actionable insights to intervene with targeted retention efforts.
- **Understand and explain how specific predictors contribute to churn risk for individual customers**. This enables personalized interventions.

Running example: H2O setup

- Go to https://h2o.ai/resources/download/
- Click on the tab **H2O Open Source Platform**.
- Go to **Latest Stable Release** or **Prior Releases**. (Stata's H2OML documentation is written using Version **3.46.0.6**).
- Click on **Download H2O**.
- After downloading the file (for example, h2o-3.46.0.11.zip), unzip it and locate the h2o.jar file.
- Place h2o.jar in a directory included in Stata's system directories (the ado-path), as returned by the **adopath** command.

Running example: data setup

- Load data into Stata's memory.

```
. use https://www.stata-press.com/data/r19/churn, clear
(Telco customer churn data)
```

- Initialize a new H2O cluster from within Stata.

```
. h2o init
```

- Store the data to an H2O frame within the cluster.

```
. _h2oframe put, into(churn)
```

- Make **churn** the current working H2O frame.

```
. _h2oframe change churn
```

Running example

```
. _h2oframe describe

          Rows:      7043
          Cols:        26
```

Column	Type	Missing	Zeros	+Inf	-Inf	Cardinality
zipcode	int	0	0	0	0	
latitude	real	0	0	0	0	
longitude	real	0	0	0	0	
tenuremonths	int	0	11	0	0	
monthlycharges	real	0	0	0	0	
totalcharges	real	11	0	0	0	
country	enum	0	7043	0	0	1
state	enum	0	7043	0	0	1
city	enum	0	4	0	0	1129
gender	enum	0	3488	0	0	2
seniorcitizen	enum	0	5901	0	0	2
partner	enum	0	3641	0	0	2
dependents	enum	0	5416	0	0	2
phoneservice	enum	0	682	0	0	2
multiplelines	enum	0	3390	0	0	3
internetserv	enum	0	2421	0	0	3
onlinesecurity	enum	0	3498	0	0	3
onlinebackup	enum	0	3088	0	0	3
deviceprotect	enum	0	3095	0	0	3
techsupport	enum	0	3473	0	0	3
streamtv	enum	0	2810	0	0	3
streammovie	enum	0	2785	0	0	3
contract	enum	0	3875	0	0	3
paperlessbill	enum	0	2872	0	0	2
paymethod	enum	0	1544	0	0	4
churn	enum	0	5174	0	0	2

Running example

```
. tabulate churn

   Churning |
     status |      Freq.     Percent        Cum.
------------+-----------------------------------
         No |      5,174       73.46       73.46
        Yes |      1,869       26.54      100.00
------------+-----------------------------------
      Total |      7,043      100.00

. global predictors latitude longitude tenuremonths monthlycharges ///
    totalcharges gender seniorcitizen partner dependents            ///
    phoneservice multiplelines internetserv onlinesecurity          ///
    onlinebackup streamtv techsupport streammovie                   ///
    contract paperlessbill paymethod deviceprotect
```

Running example

```
. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify)

Progress (%): 0 7.9 61.5 100

Gradient boosting binary classification using H2O

Response: churn
Loss:     Bernoulli
Frame:                                  Number of observations:
  Training: churn                                    Training =  7,043
                                       Cross-validation =  7,043
Cross-validation: Stratify              Number of folds     =      3

Model parameters

Number of trees      =  50              Learning rate       =     .1
              actual =  50              Learning rate decay =      1
Tree depth:                             Pred. sampling rate =      1
           Input max =   5              Sampling rate       =      1
                 min =   5              No. of bins cat.    =  1,024
                 avg = 5.0              No. of bins root    =  1,024
                 max =   5              No. of bins cont.   =     20
Min. obs. leaf split =  10              Min. split thresh.  = .00001

Metric summary
-----------------------------------------
                   |                 Cross-
            Metric |   Training  validation
-------------------+---------------------
          Log loss |   .3368515    .4022411
 Mean class error  |   .1787339    .2265423
               AUC |    .910658    .8579884
             AUCPR |    .791417    .6712718
 Gini coefficient  |    .821316    .7159769
               MSE |   .1064788    .1313416
              RMSE |   .3263109    .3624108
-----------------------------------------
```

Confusion matrix

```
. h2omlestat confmatrix

Cross-validation confusion matrix using H2O

       |      Predicted
 churn |         No        Yes |  Total  Error    Rate
-------+-----------------------+---------------------
    No |      4,153      1,021 |  5,174  1,021    .197
   Yes |        478      1,391 |  1,869    478    .256
-------+-----------------------+---------------------
 Total |      4,631      2,412 |  7,043  1,499    .213

Note: Probability threshold .3312 that maximizes F1
      metric used for classification.
```

- Use **metric**(*metric*) to select the optimal threshold.
- Use **threshold**(#) to specify the cutpoint for the predicted probabilities.
- Use **frame**(*framename*) to specify the data on which the confusion matrix is computed.

Threshold-based metrics

```
. h2omlestat threshmetric

Maximum or minimum cross-validation metrics using H2O
-----------------------------------------
              Metric |  Max/Min  Threshold
---------------------+-------------------
                  F1 |    .6498      .3312
                  F2 |    .7626      .1017
                F0.5 |    .6398      .5074
            Accuracy |    .8076      .5074
           Precision |        1      .9548
              Recall |        1      .0188
         Specificity |        1      .9548
 Min. class accuracy |    .7752      .2943
 Mean class accuracy |     .778      .2699
      True negatives |     5174      .9548
     False negatives |        0      .0188 +
      True positives |     1869      .0188
     False positives |        0      .9548 +
  True-negative rate |        1      .9548
 False-negative rate |        0      .0188 +
  True-positive rate |        1      .0188
 False-positive rate |        0      .9548 +
                 MCC |    .5089      .3312
-----------------------------------------
+ identifies minimum metrics.
```

- Use **thresholds**(*numlist*) to specify the thresholds for which to compute the performance metrics.
- Use **frame**(*framename*) to specify the data on which the performance metrics are computed.

Cross-validation summary

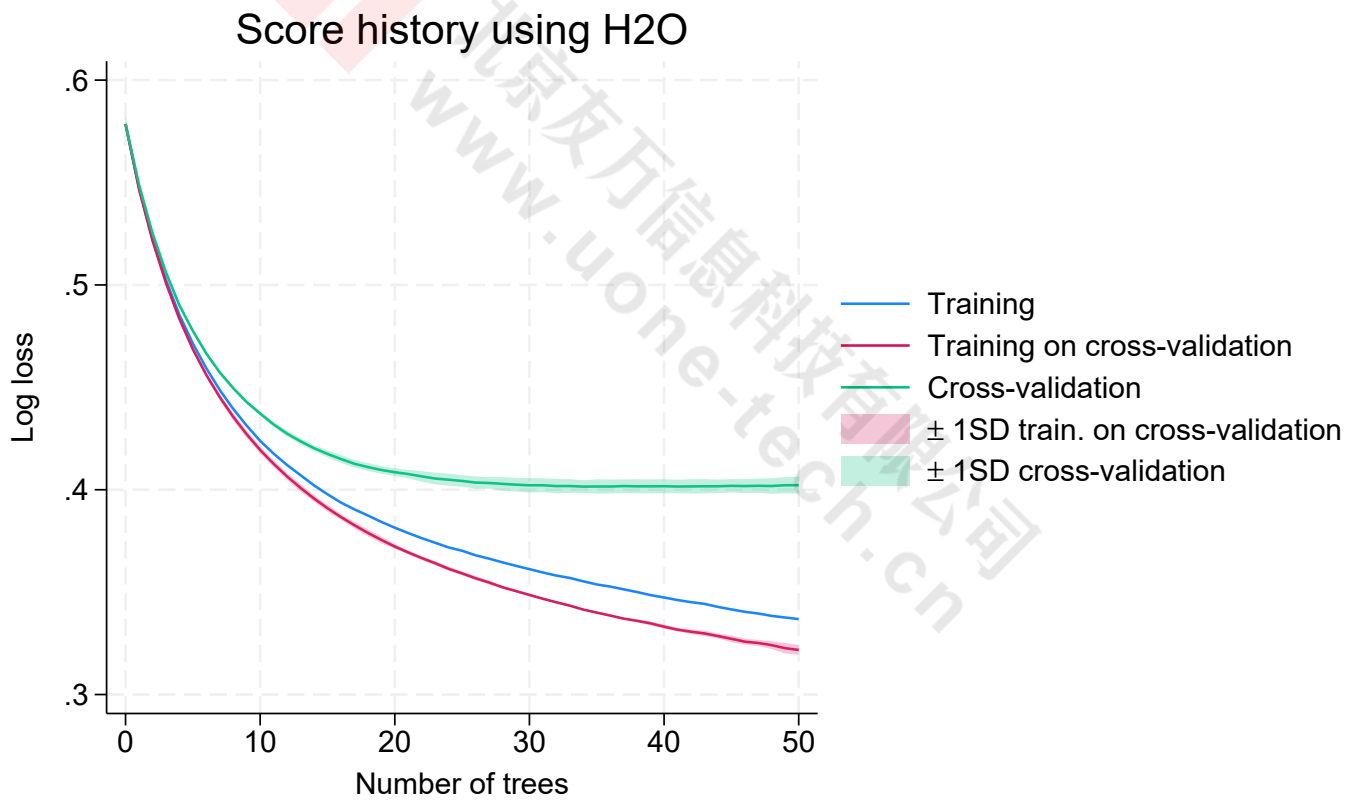
```
. h2omlestat cvsummary

Cross-validation summary using H2O
```

Metric	Mean	Std. dev.	Fold 1	Fold 2	Fold 3
Log loss	.4022452	.0050495	.4068529	.4030355	.3968471
F1	.6512654	.0115934	.6474916	.642029	.6642755
F2	.7034984	.016928	.7206671	.6868217	.7030064
F0.5	.6067057	.0211744	.5878066	.6027211	.6295893
Accuracy	.7885661	.0130515	.77469	.7904115	.8005965
Precision	.5804236	.0273417	.5537757	.579085	.60841
Recall	.743717	.0313878	.7793881	.7203252	.7314376
Specificity	.8047615	.028056	.7729918	.815155	.8261377
Misclassification	.211434	.0130515	.22531	.2095885	.1994035
Mean class error	.2257608	.0057763	.22381	.2322599	.2212124
Max. class error	.2584151	.0277609	.2270081	.2796748	.2685624
Mean class accuracy	.7742392	.0057763	.77619	.7677401	.7787877
Misclassification count	496.3333	29.56913	527	494	468
AUC	.8581837	.0055675	.8547275	.8552173	.8646063
AUCPR	.6715763	.0187176	.6638709	.6579413	.6929166
MSE	.1313421	.002305	.1329952	.132322	.128709
RMSE	.3624021	.0031875	.364685	.363761	.3587603

Score history plot

```
. h2omlgraph scorehistory
```



- Use **metric**(*metric*) to specify which metric to be plotted.

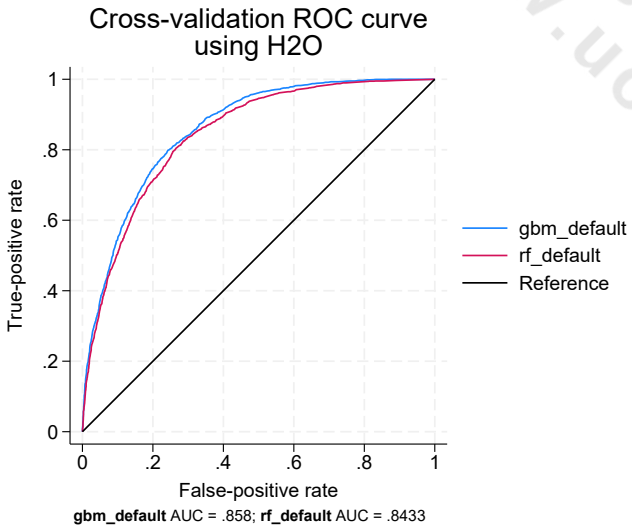
Comparison between different models

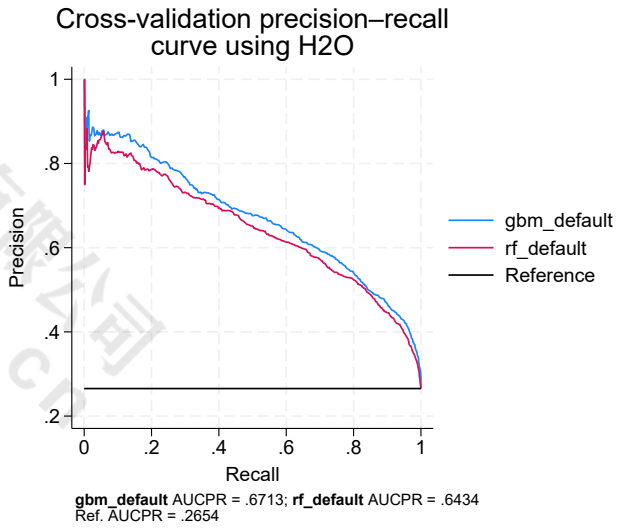
```
. h2omlest store gbm_default
. h2oml rfbinclass churn $predictors, h2orseed(19) cv(3, stratify)
  (Output omitted)
. h2omlest store rf_default
. h2omlgof gbm_default rf_default

Performance metrics for model comparison using H2O
Training frame: churn
-----------------------------------------------
                     |  gbm_def~t  rf_defa~t
---------------------+-------------------------
Training             |
  No. of observations |      7,043      7,043
           Log loss  |   .3368515   .5644957
     Mean class error |   .1787339   .2519958
                  AUC |    .910658   .8306941
               AUCPR  |    .791417   .6265946
     Gini coefficient |    .821316   .6613882
                  MSE |   .1064788   .1421843
                 RMSE |   .3263109   .3770733
---------------------+-------------------------
Cross-validation     |
  No. of observations |      7,043      7,043
           Log loss  |   .4022411   .4506645
     Mean class error |   .2265423   .2309661
                  AUC |   .8579884   .8432849
               AUCPR  |   .6712718   .6433809
     Gini coefficient |   .7159769   .6865699
                  MSE |   .1313416    .137103
                 RMSE |   .3624108   .3702743
-----------------------------------------------
```

Comparison between different models

```
. h2omlgraph roc, models(gbm_default rf_default)
. h2omlgraph prcurve, models(gbm_default rf_default)
```





Hyperparameter tuning and model selection

Hyperparameter tuning (also called *hyperparameter optimization*) is the process of systematically searching for the combination of hyperparameter values that makes a model perform best. It is used to:

- Maximize performance
- Control overfitting vs. underfitting
- Improve training efficiency

Common tuning methods:

- **Cartesian (Grid) Search** — try every combination from a predefined list of values.
- **Random Search** — sample random combinations rather than testing all of them.

Hyperparameter tuning and model selection

Hyperparameter	Description
ntrees(*numlist*)	number of trees
maxdepth(*numlist*)	maximum depth of each tree
minobsleaf(*numlist*)	minimum number of observations per child for splitting a leaf node
samprate(*numlist*)	sampling rate for randomly selecting a fraction of observations to build a tree
predsampvalue(*numlist*)	rules for how to sample predictors
minsplitthreshold(*numlist*)	threshold for the minimum relative improvement needed for a node split
and more...	

Hyperparameter tuning and model selection

Two common approaches are used for tuning hyperparameters:

- **Train/validation/test split** – split the data into training, validation, and testing sets. The validation set is used to evaluate hyperparameter choices, while the test set is reserved for final evaluation.
- **Cross-validation** – split data into training and testing sets, then use K-fold cross-validation on the training portion to evaluate hyperparameter choices.

Hyperparameter tuning and model selection

```
. _h2oframe split churn, into(train test) split(0.8 0.2)   ///
  rseed(19) replace

. _h2oframe change train
```

Grid search

```
. h2oml gbbinclass churn $predictors, h2orseed(19) cv(3, stratify) lrate(0.05) ///
>         ntrees(50(50)150) predsamprate(0.05(0.1)0.25) tune(metric(aucpr))

Progress (%): 0 100

Gradient boosting binary classification using H2O

Response: churn
Loss:     Bernoulli
Frame:                                  Number of observations:
  Training: train                                    Training =  5,643
                                       Cross-validation =  5,643
Cross-validation: Stratify              Number of folds     =      3

Tuning information for hyperparameters

Method: Cartesian
Metric: AUCPR

------------------------------------------------------------------------
                      |                    Grid values
   Hyperparameters    |       Minimum        Maximum       Selected
----------------------+-------------------------------------------------
   Number of trees    |            50             150            100
Pred. sampling rate   |           .05             .25            .15
------------------------------------------------------------------------
```

Grid search

```
Model parameters

Number of trees     = 100                 Learning rate        =    .05
              actual = 100                 Learning rate decay =      1
Tree depth:                               Pred. sampling rate =    .15
           Input max =   5                 Sampling rate        =      1
                 min =   5                 No. of bins cat.     =  1,024
                 avg = 5.0                 No. of bins root     =  1,024
                 max =   5                 No. of bins cont.    =     20
Min. obs. leaf split =  10                 Min. split thresh.   = .00001

Metric summary
-----------------------------------------
                    |                 Cross-
             Metric |   Training  validation
--------------------+--------------------
           Log loss |   .3531063    .4026141
 Mean class error   |   .1784776    .2313897
                AUC |   .8992847    .8565935
              AUCPR |   .7610732     .673929
 Gini coefficient   |   .7985693    .7131869
                MSE |   .1126847    .1314475
               RMSE |   .3356854    .3625569
-----------------------------------------

// store the best model for further analysis
. h2omlest store gbm_tuned
```

Grid search summary

```
. h2omlestat gridsummary

Grid summary using H2O
-----------------------------------
                    Pred.
     Number of   sampling
 ID      trees        rate      AUCPR
-----------------------------------
  1        100         .15   .673929
  2         50         .15  .6705027
  3        150         .15  .6702691
  4         50         .25  .6665787
  5        100         .25  .6650547
  6        150         .05  .6642718
  7        150         .25  .6606375
  8        100         .05  .6585705
  9         50         .05  .6457536
-----------------------------------
```

Model comparison and selection after grid search

```
. h2omlexplore id = 1 2 4      // Compare models

Performance metric summary using H2O
Training frame: train
------------------------------------------------------------
                     |         Model index
                     |         1          2          4
---------------------+--------------------------------------
Training             |
  No. of observations |     5,643      5,643      5,643
             Log loss |  .3531063  .3841674  .3754168
    Mean class error |  .1784776   .201682  .1942581
                 AUC |  .8992847  .8845397  .8901085
               AUCPR |  .7610732  .7302322  .7427414
    Gini coefficient |  .7985693  .7690794   .780217
                 MSE |  .1126847  .1220659  .1191758
                RMSE |  .3356854  .3493793  .3452185
---------------------+--------------------------------------
Cross-validation     |
  No. of observations |     5,643      5,643      5,643
             Log loss |  .4026141  .4111226  .4098155
    Mean class error |  .2313897  .2234232  .2210618
                 AUC |  .8565935  .8555821  .8555007
               AUCPR |   .673929  .6705027  .6665787
    Gini coefficient |  .7131869  .7111641  .7110014
                 MSE |  .1314475  .1327294  .1328578
                RMSE |  .3625569  .3643205  .3644966
------------------------------------------------------------

. h2omlselect id = 2      // Select a model as the working model
  (Output omitted)
```

Make predictions

```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)

. h2omlpostestframe test
(testing frame test is now active for h2oml postestimation)

. h2omlpredict churnhat, class
Progress (%): 0 100

. h2omlpredict churnpr1 churnpr2, pr
Progress (%): 0 100

. _h2oframe get test, clear

. list churn churnhat in 1/10

     +------------------+
     | churn   churnhat |
     |------------------|
  1. |   Yes        Yes |
  2. |   Yes        Yes |
  3. |   Yes        Yes |
  4. |   Yes        Yes |
  5. |   Yes         No |
     |------------------|
  6. |   Yes        Yes |
  7. |   Yes        Yes |
  8. |   Yes        Yes |
  9. |   Yes        Yes |
 10. |   Yes        Yes |
     +------------------+
```

Explainable machine learning

Many powerful models (deep neural networks, gradient boosting ensembles, etc.) are "**black boxes**" – it is hard to explain *why* the model made a given prediction. This matters in many real situations:

- A **loan application gets rejected** — the applicant has a legal right to know why.
- A **medical model flags a patient as high-risk** — a doctor needs to trust and verify the reasoning before acting.
- A **model behaves unexpectedly** — engineers need to debug it.

Interpretability vs. Explainability

	Interpretability	Explainability
What it means	A human can understand the model's mechanics on their own.	The model's behavior is explained after the fact, often using separate tools or techniques.
Approach	Transparent by design (white-box): you can easily trace how inputs lead to outputs.	Post-hoc methods applied to a "black box".
Typical models	Linear/logistic regression, decision trees, rule-based systems.	Neural networks, random forests, gradient boosting machine.

Explainable methods

	Local	**Global**
Scope	Explain a single, specific prediction.	Explain how the model behaves overall, across all predictions.
Techniques	SHAP (SHapley Additive exPlanations) values **h2omlgraph shapvalues** Individual Conditional Expectation (ICE) **h2omlgraph ice**	SHAP summary **h2omlgraph shapsummary** Partial Dependence Plot (PDP) **h2omlgraph pdp** Variable importance **h2omlgraph varimp** **h2omlgraph permimp**

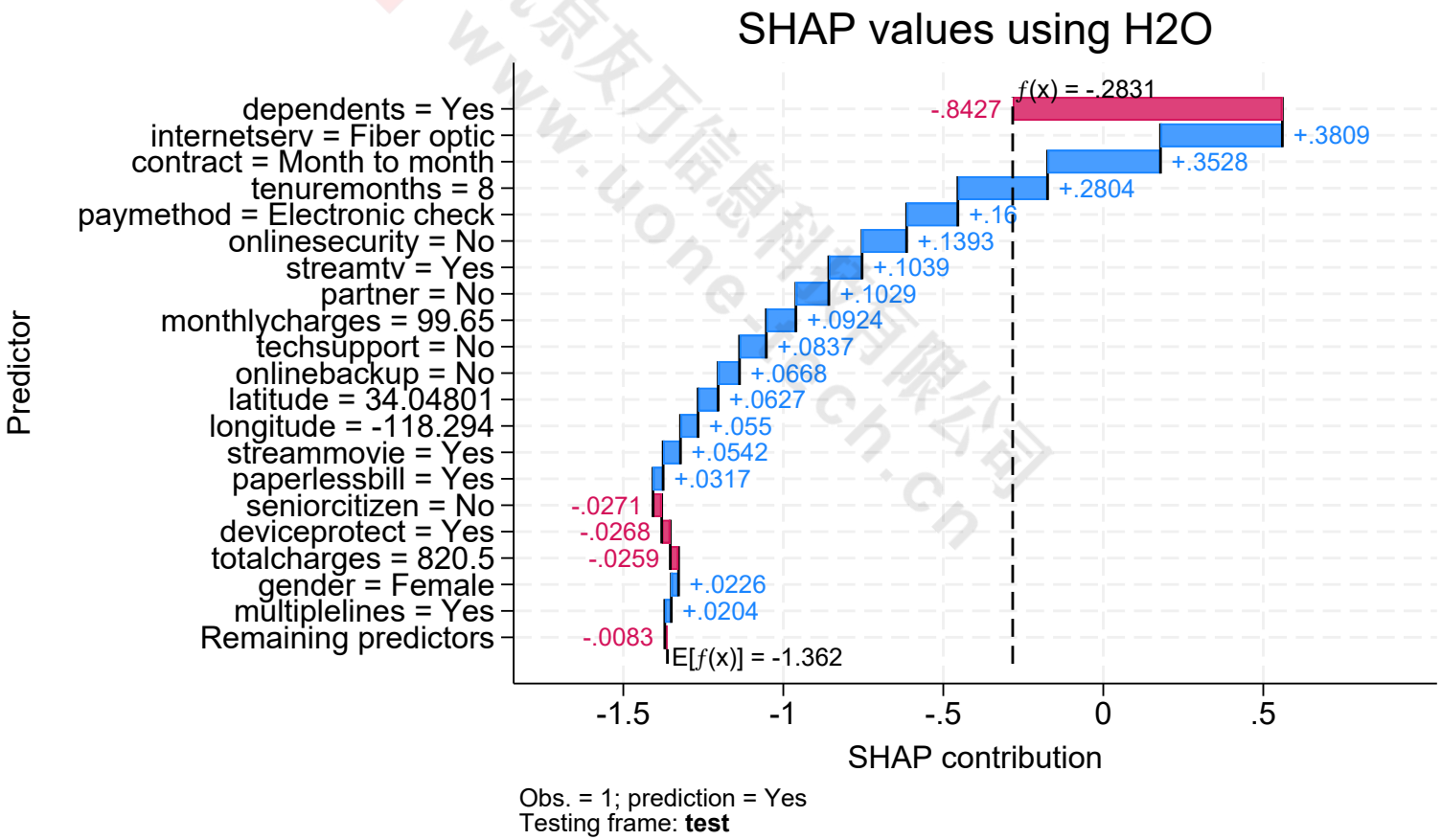
SHAP values

- A concept from cooperative game theory, developed by Lloyd Shapley in 1953.
- It tries to explain contribution of each predictor for a given observation.
- It tells how much each feature push the prediction up or down, compared to the average prediction.
- The key property is **additivity**: the average prediction plus all the feature contributions sum exactly to the final prediction.
- The result is often visualized with a **waterfall plot**.

SHAP values

```
. h2omlest restore gbm_tuned
(results gbm_tuned are active now)

. h2omlgraph shapvalues, obs(1) test(test)
```



SHAP summary

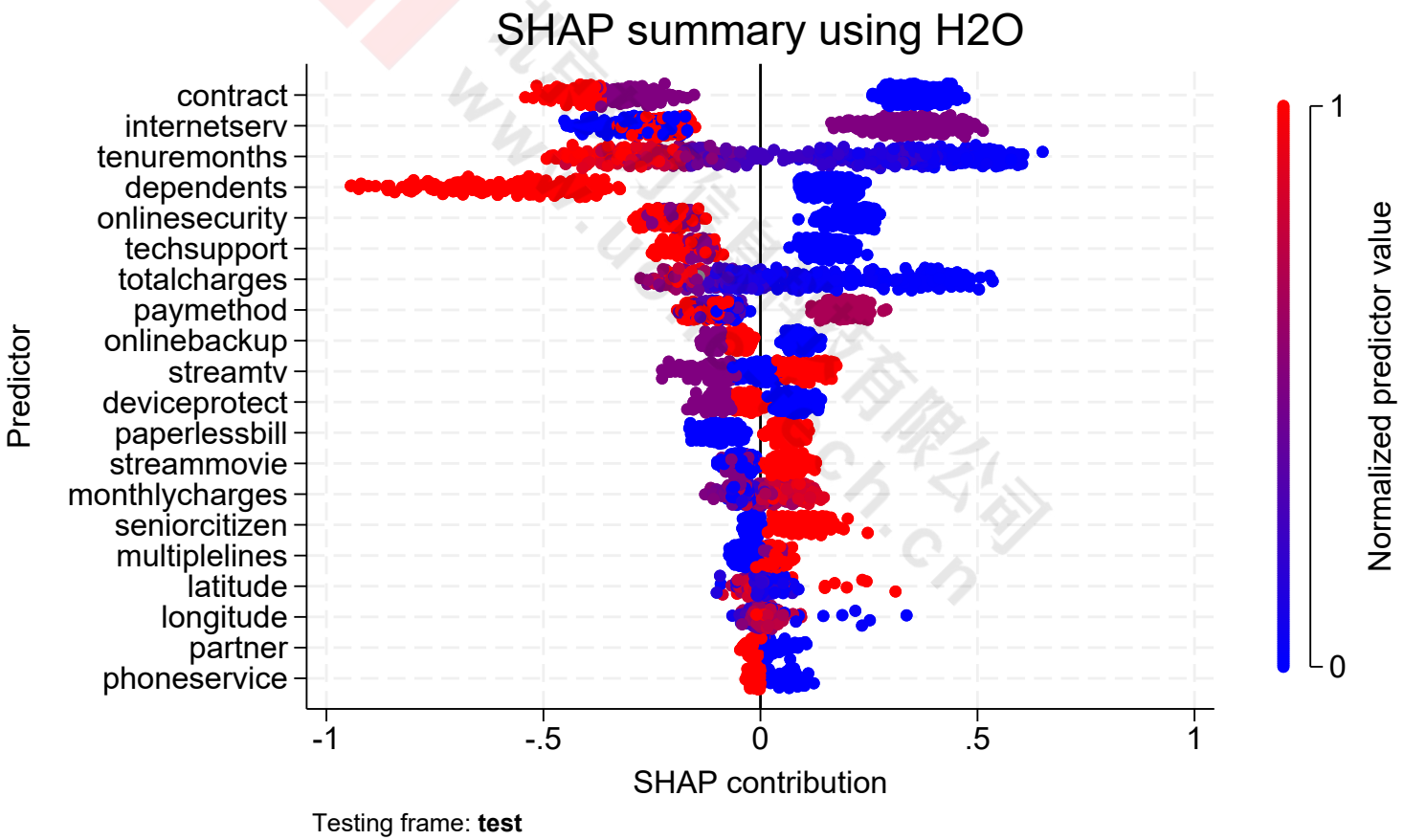
It visualizes how a model's features collectively impact its predictions by plotting every individual data point across your dataset. It reveals:

- **Which features matter most** – features are ranked by overall impact.
- **The direction of each feature's impact** – whether high values push predictions up or down.
- The spread of impact across the dataset.

This is commonly displayed as a **beeswarm plot**.

SHAP summary

```
. h2omlgraph shapsummary, rseed(19) test(test)
```



Testing frame: **test**

Partial dependency plot (PDP)

- A Partial Dependence Plot (PDP) is a global, model-agnostic explanation method that shows the marginal effect of one or two features on the predicted outcome of a machine learning model.
- It answers a specific, intuitive question: *"On average, how does the model's prediction change as I vary one feature while holding everything else constant?"*

Partial dependency plot (PDP)

Let $f(\mathbf{X}_S, \mathbf{X}_C)$ be our machine learning model, where $\mathbf{X}_S$ denotes the predictors whose effect we wish to study and $\mathbf{X}_C$ denotes all other predictors in the model. For $\mathbf{X}_S$ fixed at $\mathbf{x}_S$, the partial dependence is defined as

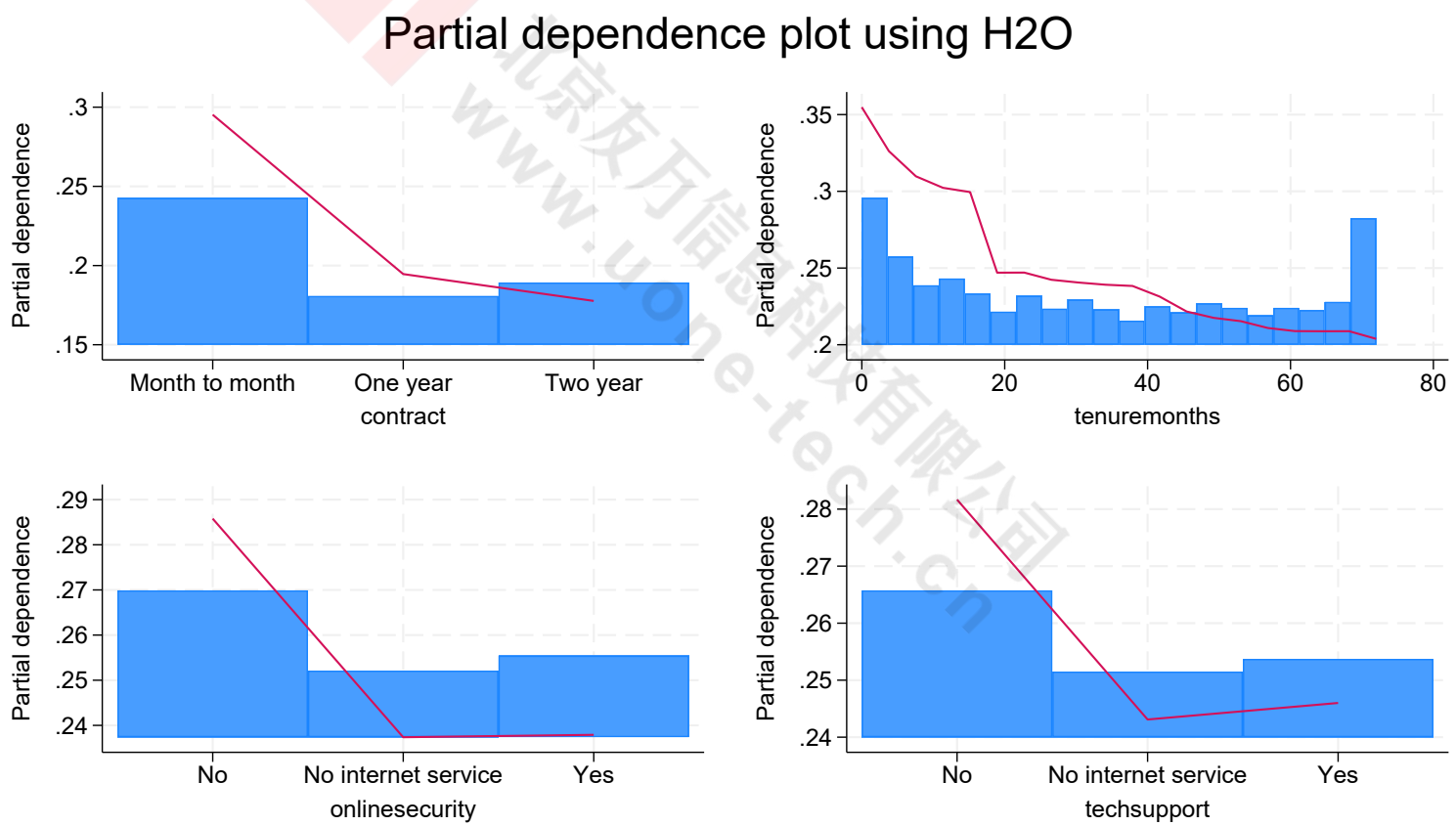
$$f_S(\mathbf{x}_S) = E_{\mathbf{X}_C}\{f(\mathbf{x}_S, \mathbf{X}_C)\} = \int f(\mathbf{x}_S, \mathbf{x}_C) dP(\mathbf{x}_C)$$

In practice, we estimate this integral by averageing the model's predictions over the actual training data. In a finite sample, the partial dependence of $\mathbf{x}_{Sj}$ is computed by averaging the predictions obtained when $\mathbf{X}_S$ is fixed at $\mathbf{x}_{Sj}$ and $\mathbf{X}_C$ takes its observed values $\mathbf{x}_{C_i}$ for $i = 1, \ldots, n$.

$$\hat{f}_S(\mathbf{x}_{Sj}) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(\mathbf{x}_{Sj}, \mathbf{x}_{C_i})$$

Partial dependency plot

```
. h2omlgraph pdp contract tenuremonths onlinesecurity techsupport, test(test) combine
```



Testing frame: **test**

Individual Conditional Expectation (ICE)

- While a PDP plots the overall *average* prediction across a population, an ICE plot unravels that average by drawing a separate line for *every single observation* in the dataset.
- It answers the question: *"How would the prediction for this specific person or row change if we varied their value for this one feature, holding everything else fixed?"*

Individual Conditional Expectation ICE)

- **For each observation** $i = 1, 2, \ldots, n$ **in the dataset:**
  - Fix the remaining predictors $\mathbf{X}_C$ at that observation's observed values $\mathbf{x}_{Ci}$.
  - **For each grid value** $j = 1, 2, \ldots, n$**:**
    - Set the target predictor $\mathbf{X}_S$ to the observed value $\mathbf{x}_{Sj}$.
    - Compute the model prediction:

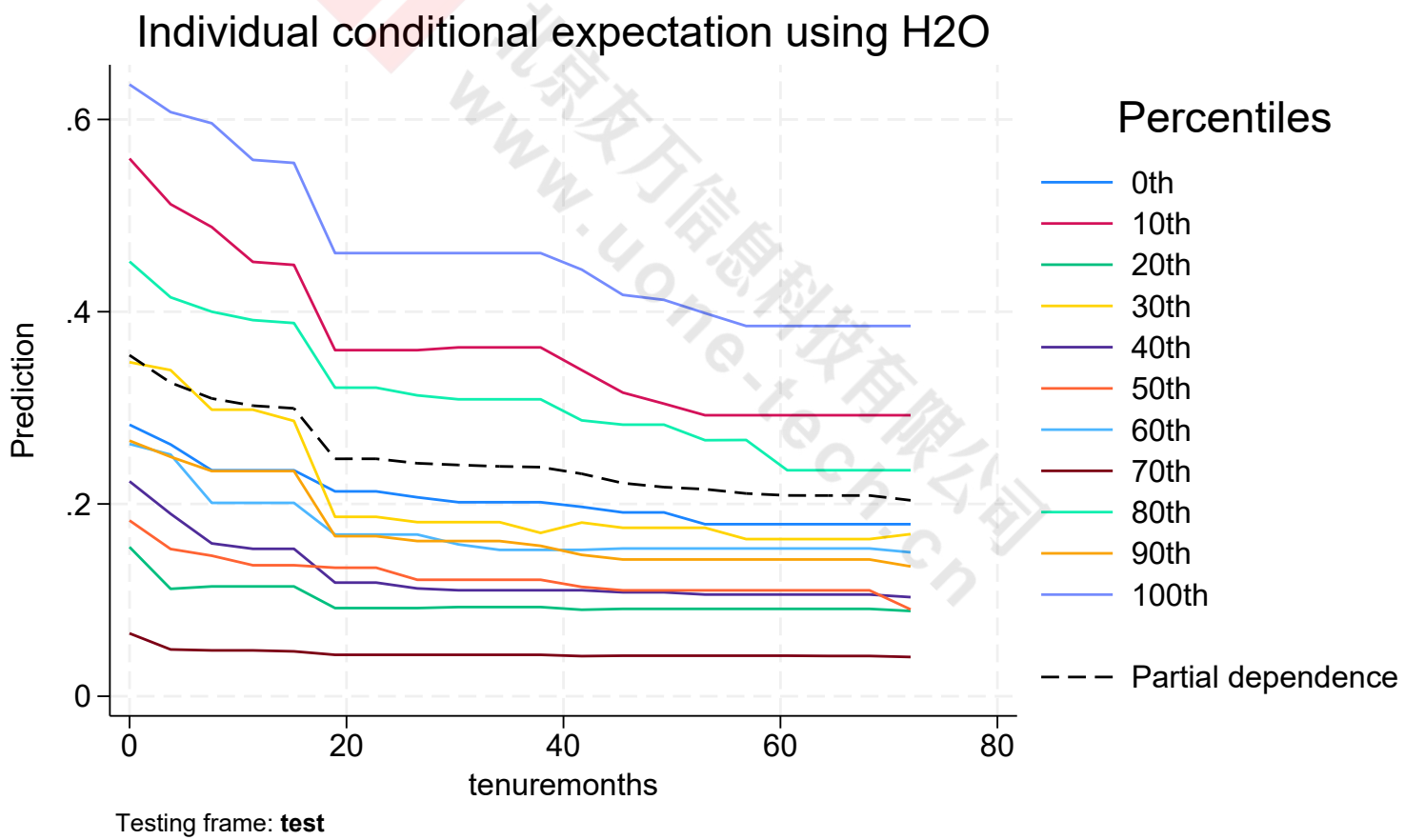
$$\widehat{f}(\mathbf{x}_{Sj}, \mathbf{x}_{Ci})$$

  - Collect the resulting set of n predictions. Together they form the conditional curve for the i-th observation.

In practice, if the number of observations is large, a graph with one curve per observation becomes difficult to read. **h2omlgraph ice** addresses this by plotting ICE curves for the deciles of the predictor of interest.

Individual Conditional Expectation

```
. h2omlgraph ice tenuremonths, test(test)
```



Testing frame: **test**

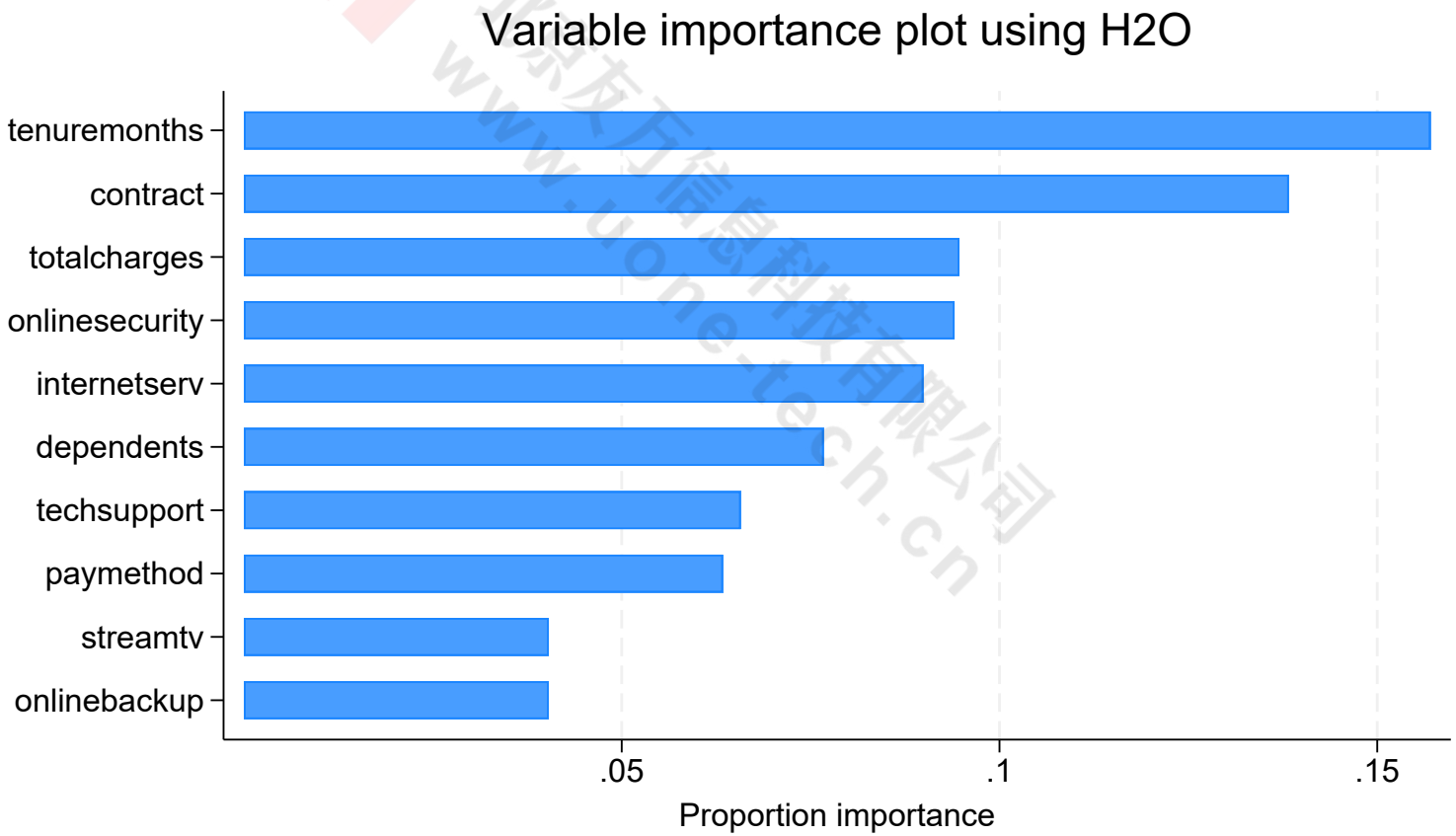
Variable importance plot

A **Variable Importance Plot** (also known as a Feature Importance Plot) is a global explanation tool that ranks features by how much they contribute to a model's predictive performance.

- **Permutation Importance (Model-Agnostic)** – shuffles the values of a single feature in the data, breaking its relationship with the target. If the model's performance drops drastically after shuffling, that feature is highly important.
- **Gini Importance / Mean Decrease in Impurity (Tree-Based)** – measures how much, on average, a feature's splits reduce impurity (for classification) or variance (for regression) across all trees in the ensemble. Features that produce larger, more frequent reduction ranks higher.

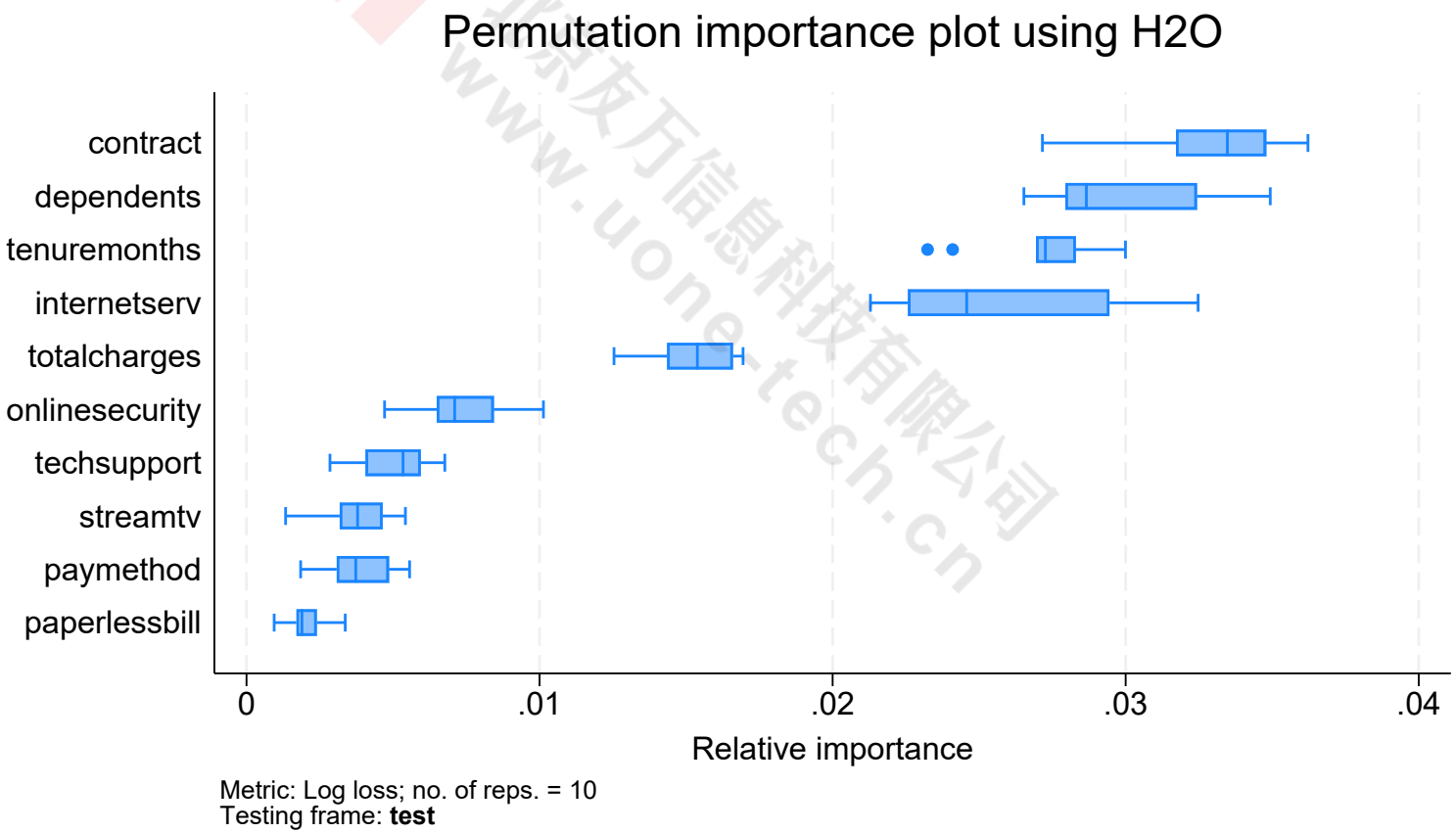
Variable importance

```
. h2omlgraph varimp
```



Variable permutation importance

```
. h2omlgraph permimp, h2orseed(19) test(test)
```



Resources

- **h2o** - Start, connect, and query an H2O cluster
  https://www.stata.com/h2o/h2o19/h2o.html
- **_h2oframe** - Work with H2O frames
  https://www.stata.com/h2o/h2o19/h2oframe.html
- **h2oml** - Machine Learning in Stata using H2O: Ensemble Decision Trees Reference Manual
  https://www.stata.com/manuals/h2oml.pdf
- **h2oml gbm** - Gradient boosting machine for regression and classification
  https://www.stata.com/manuals/h2omlh2omlgbm.pdf
- **h2oml rf** - Random forest for regression and classification
  https://www.stata.com/manuals/h2omlh2omlrf.pdf
- **h2oml postestimation** - Postestimation tools for **h2oml gbm** and **h2oml rf**
  https://www.stata.com/manuals/h2omlh2omlpostestimation.pdf

Resources

- Machine learning via H2O: Ensemble decision trees
- Stata and H2O Integration
- Dive deeper into H2O
- Approximate statistical tests for comparing binary classifier error rates using H2OML
- Prediction intervals with gradient boosting machine
- Heterogeneous treatment-effect estimation with S-, T-, and X-learners using H2OML
- H2O User Guide

Thank You!