

root

[Go Up](#)

Name	LearningTrees
Version	1.1.1
Description	LearningTrees Bundle for Tree-based Machine Learning
License	http://www.apache.org/licenses/LICENSE-2.0
Copyright	Copyright (C) 2018 HPCC Systems
Authors	HPCCSystems
DependsOn	ML_Core 3.2.0
Platform	6.4.0

OVERVIEW

LearningTrees

Bundle to provide Tree based Machine Learning algorithms.

This bundle will ultimately support all of the major Tree-based learning algorithms such as: - Decision Trees - Random Forest - Gradient Boosting Trees

The current version 1.0 supports Random Forest for classification (see `ClassificationForest.ecl`) and regression (see `RegressionForest.ecl`).

Other algorithms will be added in future versions.

Table of Contents

BoostedRegForest.ecl
Regression using Boosted Forests
ClassificationForest.ecl
Classification using Random Forest algorithm
LearningForest.ecl
This is the base module for Random Forests
LT_Types.ecl

Type definition module for Learning Trees
LUCI_Export.ecl Export a Learning Forest model to LUCI format
RegressionForest.ecl Regression using Random Forest algorithm

BoostedRegForest

[Go Up](#)

IMPORTS

```
__versions.ML_Core.V3_2_2.ML_Core |  
__versions.ML_Core.V3_2_2.ML_Core.Types | ml_core.interfaces | LT_Types |  
internal |
```

DESCRIPTIONS

BOOSTEDREGFOREST BoostedRegForest

BoostedRegForest
(UNSIGNED maxLevels=255, UNSIGNED forestSize=0, UNSIGNED maxTreeDepth=20, REAL8 learningRate=1.0, REAL8 earlyStopThreshold=.0001, SET OF UNSIGNED nominalFields=[])

Regression using Boosted Forests.

Boosted Forests (BF) are a combination of Gradient Boosted Trees (GBT) and Random Forests (RF). They provide accuracy at least as good as GBTs with the ease of use of RFs. They utilize Boosting to enhance the accuracy of Random Forests.

Layers of Random Forests are constructed, each attempting to compensate for the cumulative error of the forests before it.

A Boosted Forest with a forest size of 1 is essentially the same as a GBT. While this is supported, it is not recommended. BFs with forest size ≥ 10 have characteristics superior to both RF and GBT. They generally provide accuracy higher than RFs, and better than or equal that of expertly regularized GBTs. Yet they require no regularization, work well with default parameters, and are as easy to use as RFs.

BFs provide an early-stopping capability. This allows the number of boosting iterations (i.e. maxLevels)

to be specified at a high level (default 999), but only boosts for as many iterations as necessary to maximize accuracy. In normal practice, boosting will stop long before the default `maxLevels` is reached.

Boosted Forests share most of the benefits and limitations of Random Forests:

- Random Forests provide an effective method for regression. They are known to be one of the best out-of-the-box methods as there are few assumptions made regarding the nature of the data or its relationships.
- Random Forests can effectively manage large numbers of features, and will automatically choose the most relevant features.
- Regression Forests can handle non-linear and discontinuous relationships among features.
- A limitation of Regression Forests is that they provide no extrapolation beyond the bounds of the training data. The training set should extend to the limits of expected feature values.

This implementation allows both Ordinal (discrete or continuous) and Nominal (unordered categorical values) for the independent (X) features. There is therefore, no need to one-hot encode categorical features. Nominal features should be identified by including their feature 'number' in the set of `'nominalFields'`.

Boosted Forests support the Myriad interface meaning that multiple independent models can be computed with a single call (see `ML_Core.Types` for information on using the Myriad feature).

Notes on use of `NumericField` layouts:

- Work-item ids ('wi' field) are not required to be sequential, though they must be positive numbers. It is a good practice to assign $wi = 1$ when only one work-item is used.
- Record Ids ('id' field) are not required to be sequential, though slightly faster performance will result if they are sequential (i.e. $1 \dots \text{numRecords}$) for each work-item.
- Feature numbers ('number' field) are not required to be sequential, though slightly faster performance will result if they are (i.e. $1 \dots \text{numFeatures}$) for each work-item.

While we recommend use of the default training parameters, we do allow override of these parameters in order to attempt further regularization or optimization, or in cases where you must use straight GBTs.

Here are some guidelines for setting these parameters:

- It is not recommended to use forest sizes between 2 and 9. A forest size of 10 is the minimum to provide effective RF generalization. Forests in this range behave somewhere between GBT and BF, and will probably require regularization.
- There are three regularization parameters: `maxDepthPerTree`, `learningRate`, and `maxLevels` (i.e. the number of boosting iterations). These all interact. If early stopping is enabled (i.e. `earlyStopThreshold > 0`), then it is not necessary to regularize `maxLevels` as it will be automatically determined. For GBT (i.e. `treesPerLevel = 1`), it is necessary to regularize at least the other two parameters in order to achieve reasonable results.

- For forestSize > 9, these parameters have minimal effect, though slight gains may be possible in certain circumstances.
- For GBT (forestSize = 1), smaller sizes of maxTreeDepth (3-10) are recommended, as are low values for learningRate (< .5).
- For BF (forestSize > 9), moderate values are likely to provide optimal results: maxTreeDepth between 7 and 30 and learningRate between .5 and 1.0 might generate good results.

PARAMETER maxLevels ||| UNSIGNED8 — The maximum number of boosting iterations to perform. This is overridden by early stopping, and is primarily a failsafe in case the data is non-separable. Default (recommended) is 999.

PARAMETER forestSize ||| UNSIGNED8 — The number of trees to use in each Random Forest level. The default (recommended) is zero, which indicates that it should be automatically determined by the software.

PARAMETER maxTreeDepth ||| UNSIGNED8 — The depth to which trees are grown. Smaller numbers provide weaker learners that are needed for GBT purposes. The default (recommended) is 20.

PARAMETER learningRate ||| REAL8 — The distance along the gradient to proceed on each boosting iteration. The default (recommended) is 1.0.

PARAMETER earlyStopThreshold ||| REAL8 — A threshold against the RVR (Residual Variance Ratio) to enable early stopping. The default threshold (recommended) is .0001, which indicates that we will stop when 99.99% of the variance in the original data has been explained by the model. Setting this value to zero disables early stopping (not recommended).

PARAMETER nominalFields ||| SET (UNSIGNED8) — An optional set of field 'numbers' that represent Nominal (i.e. unordered, categorical) values. Specifying the nominal fields improves run-time performance on these fields and often improves accuracy as well. Binary fields (fields with only two values) need not be included here as they can be considered either ordinal or nominal. The default is to treat all fields as ordered. Note that this feature should only be used if all of the independent data for all work-items use the same record format, and therefore have the same set of nominal fields.

PARENT iregression2

Children

1. [Accuracy](#) : Assess the accuracy of a set of predictions
2. [GetModel](#) : Fit a model that maps independent data (X) to its prediction of (Y)
3. [Predict](#) : Predict a set of data points using a previously fitted model
4. [GetModelStats](#) : Get summary statistical information about the model
5. [Model2Nodes](#) : Extract the set of tree nodes from a model

6. FeatureImportance :

Determine the relative importance of features in the decision process of the model

ACCURACY Accuracy

BoostedRegForest \

<code>DATASET(Regression_Accuracy)</code>	Accuracy
<code>(DATASET(Layout_Model2) model, DATASET(NumericField) actuals, DATASET(NumericField) observations)</code>	

Assess the accuracy of a set of predictions. This is equivalent to calling predict and then Analysis.Regression.Accuracy.

PARAMETER model ||| TABLE (Layout_Model2) — The model as returned from GetModel

PARAMETER actuals ||| TABLE (NumericField) — The actual values of the dependent variable to compare with the predictions.

PARAMETER observations ||| TABLE (NumericField) — The independent data upon which the accuracy assessment is to be based.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 regressor , REAL8 R2 , REAL8 MSE , REAL8 RMSE }) — Accuracy statistics (see Types.Regression_Accuracy for details)

OVERRIDE

GETMODEL GetModel

BoostedRegForest \

GetModel
<code>(DATASET(NumericField) independents, DATASET(NumericField) dependents)</code>

Fit a model that maps independent data (X) to its prediction of (Y).

PARAMETER independents ||| TABLE (NumericField) — The set of independent data in NumericField format.

PARAMETER dependents ||| TABLE (NumericField) — The dependent variable in NumericField format. The 'number' field is not used as only one dependent variable is currently supported. For consistency, it should be set to 1.

RETURN TABLE (Layout_Model2) — Model in Layout_Model2 format describing the fitted forest.

SEE ML_Core.Types.NumericField, ML_Core.Types.Layout_Model2

OVERRIDE

PREDICT Predict

BoostedRegForest \

<code>DATASET(NumericField)</code>	Predict
<code>(DATASET(Layout_Model2) model, DATASET(NumericField) observations)</code>	

Predict a set of data points using a previously fitted model.

PARAMETER mod ||| — A model previously returned by GetModel in Layout_Model2 format.

PARAMETER observations ||| TABLE (NumericField) — The set of independent data in NumericField format.

PARAMETER model ||| TABLE (Layout_Model2) — No Doc

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — A NumericField dataset that provides a prediction for each record in observations.

OVERRIDE

GETMODELSTATS GetModelStats

[BoostedRegForest](#) \

DATASET (ModelStats)	GetModelStats
(DATASET(Layout_Model2) mod)	

Get summary statistical information about the model.

PARAMETER mod ||| TABLE (Layout_Model2) — A model previously returned from GetModel.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 treeCount , UNSIGNED8 minTreeDepth , UNSIGNED8 maxTreeDepth , REAL8 avgTreeDepth , UNSIGNED8 minTreeNodes , UNSIGNED8 maxTreeNodes , REAL8 avgTreeNodes , UNSIGNED8 totalNodes , UNSIGNED8 minSupport , UNSIGNED8 maxSupport , REAL8 avgSupport , REAL8 avgSupportPerLeaf , UNSIGNED8 maxSupportPerLeaf , REAL8 avgLeafDepth , UNSIGNED8 minLeafDepth , UNSIGNED8 bfLevel }) — A single ModelStats record per work-item, containing information about the model for that work-item.

SEE LT_Types.ModelStats

MODEL2NODES Model2Nodes

[BoostedRegForest](#) \

DATASET (BfTreeNodeDat)	Model2Nodes
(DATASET(Layout_Model2) mod)	

Extract the set of tree nodes from a model.

PARAMETER mod ||| TABLE (Layout_Model2) — A model as returned from GetModel.

RETURN TABLE ({ UNSIGNED4 treeId , UNSIGNED8 nodeId , UNSIGNED8 parentId , BOOLEAN isLeft , UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value , BOOLEAN isOrdinal , UNSIGNED2 level , INTEGER4 origId , REAL8 depend , UNSIGNED8 support , REAL8 ir , REAL8 observWeight , UNSIGNED2 bfLevel }) — Set of tree nodes representing the fitted forest in DATASET(TreeNodeDat) format.

SEE `LT_Types.TreeNodeDat`

FEATUREIMPORTANCE `FeatureImportance`

`BoostedRegForest \`

<code>FeatureImportance</code>
<code>(DATASET(Layout_Model2) mod)</code>

Determine the relative importance of features in the decision process of the model. Calculate feature importance using the Mean Decrease Impurity (MDI) method from "Understanding Random Forests: by Gilles Loupe (<https://arxiv.org/pdf/1407.7502.pdf>) and due to Breiman [2001, 2002].

Each feature is ranked by:

```
SUM for each branch node in which feature appears (across all trees):  
(impurity\_reduction * number of nodes split) / numTrees.
```

PARAMETER `mod` ||| `TABLE (Layout_Model2)` — The model to use for ranking of feature importance.

RETURN `TABLE ({ UNSIGNED2 wi , UNSIGNED4 number , REAL8 importance ,
UNSIGNED8 uses })` — `DATASET(FeatureImportanceRec)`, one per feature per `wi`.

SEE `LT_Types.FeatureImportanceRec`

ClassificationForest

[Go Up](#)

IMPORTS

```
LT__Types | __versions.ML_Core.V3_2_2.ML_Core |  
__versions.ML_Core.V3_2_2.ML_Core.Types | ml_core.interfaces | internal |
```

DESCRIPTIONS

CLASSIFICATIONFOREST ClassificationForest

ClassificationForest
(UNSIGNED numTrees=100, UNSIGNED featuresPerNode=0, UNSIGNED maxDepth=100, SET OF UNSIGNED nominalFields=[], BOOLEAN balanceClasses=FALSE)

Classification using Random Forest algorithm.

This module implements Random Forest classification as described by Breiman, 2001 with extensions. (see <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>)

Random Forests provide a very effective method for classification with few assumptions about the nature of the data. They are known to be one of the best out-of-the-box methods as there are few assumptions made regarding the nature of the data or its relationship to classes. Random Forests can effectively manage large numbers of features, and will automatically choose the most relevant features. Random Forests inherently support multi-class problems. Any number of class labels can be used.

This implementation supports both Numeric (discrete or continuous) and Nominal (unordered categorical values) for the independent (X) features. There is therefore, no need to one-hot encode categorical features. Nominal features should be identified by including their feature 'number' in the set of 'nominalFields' in GetModel.

RegressionForest supports the Myriad interface meaning that multiple independent models can be computed with a single call (see `ML_Core.Types` for information on using the Myriad feature).

Notes on use of `NumericField` and `DiscreteField` layouts:

- Work-item ids ('wi' field) are not required to be sequential, though they must be positive numbers. It is a good practice to assign $wi = 1$ when only one work-item is used.
- Record Ids ('id' field) are not required to be sequential, though slightly faster performance will result if they are sequential (i.e. $1 \dots numRecords$) for each work-item.
- Feature numbers ('number' field) are not required to be sequential, though slightly faster performance will result if they are (i.e. $1 \dots numFeatures$) for each work-item.

PARAMETER `numTrees` ||| UNSIGNED8 — The number of trees to create in the forest for each work-item. This defaults to 100, which is adequate for most cases. Increasing this parameter generally results in less variance in accuracy between runs, at the expense of greater run time.

PARAMETER `featuresPerNode` ||| UNSIGNED8 — The number of features to choose among at each split in each tree. This number of features will be chosen at random from the full set of features. The default (0) uses the square root of the number of features provided, which works well for most cases.

PARAMETER `maxDepth` ||| UNSIGNED8 — The deepest to grow any tree in the forest. The default is 100, which is adequate for most purposes. Increasing this value for very large and complex problems may provide slightly greater accuracy at the expense of much greater runtime.

PARAMETER `nominalFields` ||| SET (UNSIGNED8) — An optional set of field 'numbers' that represent Nominal (i.e. unordered, categorical) values. Specifying the nominal fields improves run-time performance on these fields and may improve accuracy as well. Binary fields (fields with only two values) need not be included here as they can be considered either ordinal or nominal. The default is to treat all fields as ordered. Note that this feature should only be used if all of the independent data for all work-items use the same record format, and therefore have the same set of nominal fields.

PARAMETER `balanceClasses` ||| BOOLEAN — An optional Boolean parameter. If true, it indicates that the voting among trees should be biased inversely to the frequency of the class for which it is voting. This may help in scenarios where there are far more samples of certain classes than of others. The default is to not balance (i.e. FALSE).

PARENT `LearningForest`

PARENT `iclassify2`

Children

1. [GetModelStats](#) : Get summary statistical information about the model
2. [Model2Nodes](#) : Extract the set of tree nodes from a model

3. [Accuracy](#) : Return accuracy metrics for the given set of test data
This is equivalent to calling Predict followed by Analysis.Classification.Accuracy(...)
4. [FeatureImportance](#) :
Determine the relative importance of features in the decision process of the model
5. [AccuracyByClass](#) : Return class-level accuracy by class metrics for the given set of test data
6. [GetModel](#) : Fit and return a model that maps independent data (X) to its predicted class (Y)
7. [Classify](#) : Classify a set of data points using a previously fitted model
8. [DecisionDistanceMatrix](#) :
Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS)
9. [ConfusionMatrix](#) : Return the confusion matrix for a set of test data
10. [GetClassProbs](#) : Calculate the 'probability' that each data point is in each class
11. [Model2ClassWeights](#) : Extract the set of class weights from the model
12. [UniquenessFactor](#) : Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points
13. [CompressModel](#) : Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child

GETMODELSTATS **GetModelStats**

[ClassificationForest](#) \

DATASET(ModelStats)	GetModelStats
(DATASET(Layout_Model2) mod)	

Get summary statistical information about the model.

PARAMETER **mod** ||| TABLE (Layout_Model2) — A model previously returned from GetModel.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 treeCount , UNSIGNED8 minTreeDepth , UNSIGNED8 maxTreeDepth , REAL8 avgTreeDepth , UNSIGNED8 minTreeNodees , UNSIGNED8 maxTreeNodees , REAL8 avgTreeNodees , UNSIGNED8 totalNodes , UNSIGNED8 minSupport , UNSIGNED8 maxSupport , REAL8 avgSupport , REAL8 avgSupportPerLeaf , UNSIGNED8 maxSupportPerLeaf , REAL8 avgLeafDepth , UNSIGNED8 minLeafDepth , UNSIGNED8 bfLevel }) — A single ModelStats record per work-item, containing information about the model for that work-item.

SEE `LT_Types.ModelStats`

INHERITED

MODEL2NODES `Model2Nodes`

`ClassificationForest` \

<code>DATASET(TreeNodeDat)</code>	<code>Model2Nodes</code>
<code>(DATASET(Layout_Model2) mod)</code>	

Extract the set of tree nodes from a model.

PARAMETER `mod` ||| `TABLE (Layout_Model2)` — A model as returned from `GetModel`.

RETURN `TABLE ({ UNSIGNED4 treeId , UNSIGNED8 nodeId , UNSIGNED8 parentId , BOOLEAN isLeft , UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value , BOOLEAN isOrdinal , UNSIGNED2 level , INTEGER4 origId , REAL8 depend , UNSIGNED8 support , REAL8 ir , REAL8 observWeight })` — Set of tree nodes representing the fitted forest in `DATASET(TreeNodeDat)` format.

SEE `LT_Types.TreeNodeDat`

INHERITED

ACCURACY `Accuracy`

`ClassificationForest` \

<code>DATASET(Classification_Accuracy)</code>	<code>Accuracy</code>
<code>(DATASET(Layout_Model2) model, DATASET(DiscreteField) actuals, DATASET(NumericField) observations)</code>	

Return accuracy metrics for the given set of test data

This is equivalent to calling Predict followed by Analysis.Classification.Accuracy(...).

Provides accuracy statistics as follows:

- **errCnt** – The number of misclassified samples.
- **errPct** – The percentage of samples that were misclassified (0.0 - 1.0).
- **RawAccuracy** – The percentage of samples properly classified (0.0 - 1.0).
- **PoD** – Power of Discrimination. Indicates how this classification performed relative to a random guess of class. Zero or negative indicates that the classification was no better than a random guess. 1.0 indicates a perfect classification. For example if there are two equiprobable classes, then a random guess would be right about 50% of the time. If this classification had a Raw Accuracy of 75%, then its PoD would be .5 (half way between a random guess and perfection).
- **PoDE** – Power of Discrimination Extended. Indicates how this classification performed relative to guessing the most frequent class (i.e. the trivial solution). Zero or negative indicates that this classification is no better than the trivial solution. 1.0 indicates perfect classification. For example, if 95% of the samples were of class 1, then the trivial solution would be right 95% of the time. If this classification had a raw accuracy of 97.5%, its PoDE would be .5 (i.e. half way between trivial solution and perfection).

Normally, this should be called using data samples that were not included in the training set. In that case, these statistics are considered Out-of-Sample error statistics. If it is called with the X and Y from the training set, it provides In-Sample error statistics, which should never be used to rate the classification model.

PARAMETER model ||| TABLE (Layout_Model2) — The encoded model as returned from GetModel.

PARAMETER actuals ||| TABLE (DiscreteField) — The actual class values associated with the observations.

PARAMETER observations ||| TABLE (NumericField) — The independent (explanatory) values on which to base the test.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 classifier , UNSIGNED8 recCnt , UNSIGNED8 errCnt , REAL8 Raw_Accuracy , REAL8 PoD , REAL8 PoDE , REAL8 Hamming_Loss }) — DATSET(Classification_Accuracy), one record per work-item.

SEE Types.Classification_Accuracy

OVERRIDE

FEATUREIMPORTANCE FeatureImportance

ClassificationForest \

	FeatureImportance
(DATASET(Layout_Model2) mod)	

Determine the relative importance of features in the decision process of the model. Calculate feature importance using the Mean Decrease Impurity (MDI) method from "Understanding Random Forests: by Gilles Loupe (<https://arxiv.org/pdf/1407.7502.pdf>) and due to Breiman [2001, 2002].

Each feature is ranked by:

```
SUM for each branch node in which feature appears (across all trees):  
(impurity\_reduction * number of nodes split) / numTrees.
```

PARAMETER mod ||| TABLE (Layout_Model2) — The model to use for ranking of feature importance.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 number , REAL8 importance , UNSIGNED8 uses }) — DATASET(FeatureImportanceRec), one per feature per wi.

SEE LT_Types.FeatureImportanceRec

INHERITED

ACCURACYBYCLASS AccuracyByClass

ClassificationForest \

DATASET(Class_Accuracy)	AccuracyByClass
(DATASET(Layout_Model2) model, DATASET(DiscreteField) actuals, DATASET(NumericField) observations)	

Return class-level accuracy by class metrics for the given set of test data.

This is equivalent to calling Predict followed by Analysis.Classification.AccuracyByClass(...).

PARAMETER model ||| TABLE (Layout_Model2) — The encoded model as returned from GetModel.

PARAMETER actuals ||| TABLE (DiscreteField) — The actual class values associated with the observations.

PARAMETER observations ||| TABLE (NumericField) — The independent (explanatory) values on which to base the test

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 classifier , INTEGER4 class , REAL8 precision , REAL8 recall , REAL8 FPR , REAL8 f_score }) — DATASET(Class_Accuracy), one record per work-item per class.

SEE Types.Class_Accuracy.

OVERLOAD

GETMODEL GetModel

[ClassificationForest](#) \

DATASET(Layout_Model2)	GetModel
(DATASET(NumericField) independents, DATASET(DiscreteField) dependents)	

Fit and return a model that maps independent data (X) to its predicted class (Y).

PARAMETER independents ||| TABLE (NumericField) — The set of independent data in NumericField format.

PARAMETER dependents ||| TABLE (DiscreteField) — The set of classes in DiscreteField format that correspond to the independent data i.e. same 'id'.

PARAMETER nominalFields ||| — An optional set of field 'numbers' that represent Nominal (i.e. unordered, categorical) values. Specifying the nominal fields improves run-time performance on these fields and may improve accuracy as well. Binary fields (fields with only two values) need not be listed here as they can be considered either ordinal or nominal. Example: [3,5,7].

RETURN TABLE ({ UNSIGNED2 wi , REAL8 value , SET (UNSIGNED4) indexes }) — Model in Layout_Model2 format describing the fitted forest.

SEE `ML_Core.Types.NumericField`, `ML_Core.Types.DiscreteField`, `ML_Core.Types.Layout_Model2`

OVERRIDE

CLASSIFY `Classify`

`ClassificationForest` \

<code>DATASET(DiscreteField)</code>	<code>Classify</code>
<code>(DATASET(Layout_Model2) model, DATASET(NumericField) observations)</code>	

Classify a set of data points using a previously fitted model

PARAMETER `model` ||| `TABLE (Layout_Model2)` — A model previously returned by `GetModel` in `Layout_Model2` format.

PARAMETER `observations` ||| `TABLE (NumericField)` — The set of independent data to classify in `NumericField` format.

RETURN `TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , INTEGER4 value })` — A `DiscreteField` dataset that indicates the predicted class of each item in `observations`.

OVERRIDE

DECISIONDISTANCEMATRIX `DecisionDistanceMatrix`

`ClassificationForest` \

	<code>DecisionDistanceMatrix</code>
<code>(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)</code>	

Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS). This is an experimental method and may not scale to large numbers of data point combinations. Two sets of data points X1 and X2 are taken as parameters. A Decision Distance will be returned for every point in X1 to every point in X2. Therefore, if X1 has N points and X2 has M points, an N x M matrix of results will be produced. X2 may be omitted, in which case, an N x N matrix will be produced with a Decision Distance for every pair of points in X1.

This metric represents a distance measure in the RFDS. As such, it provides a continuous measure of distance in a space that is highly non-linear and discontinuous relative to the training data. Distances in RFDS can be thought of as the number of binary decisions that separate two points in the tree. DD, however is a normalized metric $0 \leq DD < 1$ that incorporates the depth of the decision tree. It is also averaged over all of the trees in the forest. It can possibly be viewed as an approximation of the relative Hamming Distances between points.

PARAMETER **mod** ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER **X1** ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER **X2** ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'number' is the id of the "to" point. 'value' contains the DD metric between "from" and "to" points. Note that if the same point is in X1 and X2, there will be redundant metrics, since DD is a symmetric measure (i.e. $DD(x1, x2) = DD(x2, x1)$).

INHERITED

CONFUSIONMATRIX ConfusionMatrix

ClassificationForest \

DATASET(Confusion_Detail)	ConfusionMatrix
(DATASET(Layout_Model2) model, DATASET(DiscreteField) actuals, DATASET(NumericField) observations)	

Return the confusion matrix for a set of test data. This is equivalent to calling Predict followed by Analysis.Classification.ConfusionMatrix(...).

The confusion matrix indicates the number of datapoints that were classified correctly or incorrectly for each class label.

The matrix is provided as a matrix of size numClasses x numClasses with fields as follows:

- 'wi' – The work item id
- 'pred' – the predicted class label (from Classify).
- 'actual' – the actual (target) class label.
- 'samples' – the count of samples that were predicted as 'pred', but should have been 'actual'.
- 'totSamples' – the total number of samples that were predicted as 'pred'.
- 'pctSamples' – the percentage of all samples that were predicted as 'pred', that should have been 'actual' (i.e. samples / totSamples)

This is a useful tool for understanding how the algorithm achieved the overall accuracy. For example: were the common classes mostly correct, while less common classes often misclassified? Which classes were most often confused? This should be called with test data that is independent of the training data in order to understand the out-of-sample (i.e. generalization) performance.

PARAMETER model ||| TABLE (Layout_Model2) — The encoded model as returned from GetModel.

PARAMETER actuals ||| TABLE (DiscreteField) — The actual class values.

PARAMETER observations ||| TABLE (NumericField) — The independent (explanatory) values.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 classifier , INTEGER4 actual_class , INTEGER4 predict_class , UNSIGNED4 occurs , BOOLEAN correct , REAL8 pctActual , REAL8 pctPred }) — DATASET(Confusion_Detail), one record per cell of the confusion matrix.

SEE Types.Confusion_Detail.

OVERRIDE

GETCLASSPROBS GetClassProbs

ClassificationForest \

DATASET(ClassProbs)	GetClassProbs
(DATASET(Layout_Model2) model, DATASET(NumericField) observations)	

Calculate the 'probability' that each data point is in each class.

Probability is approximated by computing the proportion of trees that voted for each class for each data point, so should not be treated as a reliable measure of true probability.

PARAMETER model ||| TABLE (Layout_Model2) — A model previously returned by GetModel in Layout_Model2 format.

PARAMETER observations ||| TABLE (NumericField) — The set of independent data to classify in NumericField format.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , INTEGER4 class , INTEGER4 cnt , REAL8 prob }) — DATASET(ClassProbs), one record per datapoint (i.e. id) per class label. Class labels with zero votes are omitted.

SEE LT_Types.ClassProbs

MODEL2CLASSWEIGHTS Model2ClassWeights

[ClassificationForest](#) \

Model2ClassWeights
(DATASET(Layout_Model2) mod)

Extract the set of class weights from the model.

Classes are weighted inversely proportional to their frequency in the training data.

Note that the class weights are based on a non-linear 'proportion' to avoid excess weight for classes with very low frequency.

These weights are only used when the 'balanceClasses' option is TRUE.

PARAMETER mod ||| TABLE (Layout_Model2) — A model as returned from GetModel.

RETURN TABLE ({ UNSIGNED2 wi , INTEGER4 classLabel , REAL8 weight }) — DATASET(ClassWeightRec) representing weight for each class label.

SEE LT_Types.ClassWeightRec

UNIQUENESSFACTOR UniquenessFactor

[ClassificationForest](#) \

UniquenessFactor
(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)

Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points. It may not scale to large numbers of data points. Uniqueness Factor looks at the Decision Distance from each point to every other point in a set. It is similar to Decision Distance (above), but rather than providing a distance of each "from" point to every "to" point, it provides the average distance of each "from" point to all of the "to" points. Like Decision Distance, UF lies on the interval: $0 \leq UF < 1$. A high value of UF may indicate an anomalous data point, while a low value may indicate "typicalness" of a data point. It may therefore have utility for anomaly detection or conversely, for the identification of class prototypes (e.g. the members of a class with the lowest UF). In a two-step process one could potentially compute class prototypes and then look at the distance of a point from all class prototypes. This could result in a way to detect anomalies with respect to e.g., known usage patterns.

PARAMETER **mod** ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER **X1** ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER **X2** ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'value' contains the UF metric for the point. I.e. the average DD of the "from" point to all "to" points. The 'number' field is not used and is set to 1.

INHERITED

COMPRESSMODEL CompressModel

[ClassificationForest](#) \

CompressModel
(DATASET(Layout_Model2) mod)

Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child. These branches are a result of the RF algorithm, and do not affect the results of the model. This is an expensive operation, which is why it is not done as a matter of course. It reduces the size of the model somewhat, and therefore slightly speeds up any processing that uses the model, and reduces storage size. You may want to compress the model if storage is at a premium, or if the model is to be used many times (so that the slight performance gain is multiplied). This also makes the model somewhat more readable, and could be useful when analyzing the tree or converting it to another system (e.g. LUCI) for processing.

PARAMETER **mod** ||| TABLE (Layout_Model2) — Model as returned from GetModel in Layout_Model2 format.

RETURN TABLE ({ UNSIGNED2 wi , REAL8 value , SET (UNSIGNED4) indexes })
— The Compressed Model.

SEE ML_Core.Types.Layout_Model2

INHERITED

LearningForest

[Go Up](#)

IMPORTS

```
LT_Types | __versions.ML_Core.V3_2_2.ML_Core |  
__versions.ML_Core.V3_2_2.ML_Core.Types | internal |
```

DESCRIPTIONS

LEARNINGFOREST LearningForest

LearningForest
(UNSIGNED numTrees=100, UNSIGNED featuresPerNode=0, UNSIGNED maxDepth=100)

This is the base module for Random Forests. It implements the Random Forest algorithms as described by Breiman, 2001 (see <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>).

PARAMETER numTrees ||| UNSIGNED8 — The number of trees to create as the forest for each work-item. This defaults to 100, which is adequate for most cases.

PARAMETER featuresPerNode ||| UNSIGNED8 — The number of features to choose among at each split in each tree. This number of features will be chosen at random from the full set of features. The default is the square root of the number of features provided, which works well for most cases.

PARAMETER maxDepth ||| UNSIGNED8 — The deepest to grow any tree in the forest. The default is 100, which is adequate for most purposes. Increasing this value for very large and complex problems may provide slightly greater accuracy at the expense of much greater runtime.

Children

1. [GetModelStats](#) : Get summary statistical information about the model

2. [Model2Nodes](#) : Extract the set of tree nodes from a model
3. [FeatureImportance](#) :
Determine the relative importance of features in the decision process of the model
4. [DecisionDistanceMatrix](#) :
Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS)
5. [UniquenessFactor](#) : Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points
6. [CompressModel](#) : Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child

GETMODELSTATS GetModelStats

[LearningForest](#) \

DATASET (ModelStats)	GetModelStats
(DATASET (Layout_Model2) mod)	

Get summary statistical information about the model.

PARAMETER mod ||| [TABLE](#) ([Layout_Model2](#)) — A model previously returned from [GetModel](#).

RETURN [TABLE](#) ({ [UNSIGNED2](#) wi , [UNSIGNED8](#) treeCount , [UNSIGNED8](#) minTreeDepth , [UNSIGNED8](#) maxTreeDepth , [REAL8](#) avgTreeDepth , [UNSIGNED8](#) minTreeNodes , [UNSIGNED8](#) maxTreeNodes , [REAL8](#) avgTreeNodes , [UNSIGNED8](#) totalNodes , [UNSIGNED8](#) minSupport , [UNSIGNED8](#) maxSupport , [REAL8](#) avgSupport , [REAL8](#) avgSupportPerLeaf , [UNSIGNED8](#) maxSupportPerLeaf , [REAL8](#) avgLeafDepth , [UNSIGNED8](#) minLeafDepth , [UNSIGNED8](#) bfLevel }) — A single [ModelStats](#) record per work-item, containing information about the model for that work-item.

SEE [LT_Types.ModelStats](#)

MODEL2NODES Model2Nodes

LearningForest \

DATASET(TreeNodeDat)	Model2Nodes
(DATASET(Layout_Model2) mod)	

Extract the set of tree nodes from a model.

PARAMETER mod ||| TABLE (Layout_Model2) — A model as returned from GetModel.

RETURN TABLE ({ UNSIGNED4 treeId , UNSIGNED8 nodeId , UNSIGNED8 parentId , BOOLEAN isLeft , UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value , BOOLEAN isOrdinal , UNSIGNED2 level , INTEGER4 origId , REAL8 depend , UNSIGNED8 support , REAL8 ir , REAL8 observWeight }) — Set of tree nodes representing the fitted forest in DATASET(TreeNodeDat) format.

SEE LT_Types.TreeNodeDat

FEATUREIMPORTANCE FeatureImportance

LearningForest \

FeatureImportance
(DATASET(Layout_Model2) mod)

Determine the relative importance of features in the decision process of the model. Calculate feature importance using the Mean Decrease Impurity (MDI) method from "Understanding Random Forests: by Gilles Loupe (<https://arxiv.org/pdf/1407.7502.pdf>) and due to Breiman [2001, 2002].

Each feature is ranked by:

```
SUM for each branch node in which feature appears (across all trees):  
(impurity\_reduction * number of nodes split) / numTrees.
```

PARAMETER mod ||| TABLE (Layout_Model2) — The model to use for ranking of feature importance.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 number , REAL8 importance , UNSIGNED8 uses }) — DATASET(FeatureImportanceRec), one per feature per wi.

SEE LT_Types.FeatureImportanceRec

DECISIONDISTANCEMATRIX DecisionDistanceMatrix

[LearningForest](#) \

DecisionDistanceMatrix
(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)

Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS). This is an experimental method and may not scale to large numbers of data point combinations. Two sets of data points X1 and X2 are taken as parameters. A Decision Distance will be returned for every point in X1 to every point in X2. Therefore, if X1 has N points and X2 has M points, an N x M matrix of results will be produced. X2 may be omitted, in which case, an N x N matrix will be produced with a Decision Distance for every pair of points in X1.

This metric represents a distance measure in the RFDS. As such, it provides a continuous measure of distance in a space that is highly non-linear and discontinuous relative to the training data. Distances in RFDS can be thought of as the number of binary decisions that separate two points in the tree. DD, however is a normalized metric $0 \leq DD < 1$ that incorporates the depth of the decision tree. It is also averaged over all of the trees in the forest. It can possibly be viewed as an approximation of the relative Hamming Distances between points.

PARAMETER mod ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER X1 ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER X2 ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'number' is the id of the "to" point. 'value' contains the DD metric between "from" and "to" points. Note that if the same point is in X1 and X2, there will be redundant metrics, since DD is a symmetric measure (i.e. $DD(x1, x2) = DD(x2, x1)$).

UNIQUENESSFACTOR UniquenessFactor

[LearningForest](#) \

UniquenessFactor
<code>(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)</code>

Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points. It may not scale to large numbers of data points. Uniqueness Factor looks at the Decision Distance from each point to every other point in a set. It is similar to Decision Distance (above), but rather than providing a distance of each "from" point to every "to" point, it provides the average distance of each "from" point to all of the "to" points. Like Decision Distance, UF lies on the interval: $0 \leq UF < 1$. A high value of UF may indicate an anomalous data point, while a low value may indicate "typicalness" of a data point. It may therefore have utility for anomaly detection or conversely, for the identification of class prototypes (e.g. the members of a class with the lowest UF). In a two-step process one could potentially compute class prototypes and then look at the distance of a point from all class prototypes. This could result in a way to detect anomalies with respect to e.g., known usage patterns.

PARAMETER **mod** ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER **X1** ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER **X2** ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'value' contains the UF metric for the point. I.e. the average DD of the "from" point to all "to" points. The 'number' field is not used and is set to 1.

COMPRESSMODEL CompressModel

[LearningForest](#) \

CompressModel
<code>(DATASET(Layout_Model2) mod)</code>

Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child. These branches are a result of the RF algorithm, and do not affect the results of the model. This is an expensive operation, which is why it is not done as a matter of course. It reduces the size of the model somewhat, and therefore slightly speeds up any processing that uses the model, and reduces storage size. You may want to compress the model if storage is at a premium, or if the model is to be used many times (so that the slight performance gain is multiplied). This also makes the model somewhat more readable, and could be useful when analyzing the tree or converting it to another system (e.g. LUCI) for processing.

PARAMETER **mod** ||| TABLE (Layout_Model2) — Model as returned from GetModel in Layout_Model2 format.

RETURN TABLE ({ UNSIGNED2 wi , REAL8 value , SET (UNSIGNED4) indexes })
— The Compressed Model.

SEE ML_Core.Types.Layout_Model2

LT__Types

[Go Up](#)

IMPORTS

```
__versions.ML_Core.V3__2__2.ML_Core |  
__versions.ML_Core.V3__2__2.ML_Core.Types |
```

DESCRIPTIONS

LT_TYPES LT__Types

LT__Types

Type definition module for Learning Trees.

Children

1. [t_NodeId](#) : Type definition for the node id field representing a tree node's id
2. [Forest_Model](#) : Definition of the meaning of the indexes of the Forest Model variables
3. [Bf_Model](#) : Definition of the meaning of the indexes of the Gradient Boosting Model variables
4. [GenField](#) : GenField extends NumericField by adding an isOrdinal field
5. [TreeNodeDat](#) :
This is the major working structure for building the forest
6. [BfTreeNodeDat](#) : Main data structure for processing Boosted Forest
7. [ClassProbs](#) : The probability that a given sample is of a given class
8. [NodeSummary](#) : NodeSummary provides information to identify a given tree node

9. [SplitDat](#) : SplitDat is used to hold information about a potential split
10. [NodeImpurity](#) : NodeImpurity carries identifying information for a node as well as its impurity level It is based on the NodeSummary record type above, but includes an assessment of the 'impurity' of the data at this node (i.e
11. [wiInfo](#) : Provides a summary of each work item for use in building the forest
12. [ModelStats](#) : Model Statistics Record
13. [FeatureImportanceRec](#) : Feature Importance Record describes the importance of each feature
14. [ClassWeightsRec](#) : ClassWeightsRecord holds the weights associated with each class label
15. [LUCI_Scorecard](#) : Structure used to describe the Scorecards for LUCI format export

T_NODEID t_NodeId

[LT_Types](#) \

	t_NodeId
--	-----------------

Type definition for the node id field representing a tree node's id

RETURN UNSIGNED8 —

FOREST_MODEL Forest_Model

[LT_Types](#) \

	Forest_Model
--	---------------------

Definition of the meaning of the indexes of the Forest Model variables.

Ind1 enumerates the first index, which is used to determine which type of data is stored:

- nodes stores the list of tree nodes that describes the forest. The second index is just the sequential number of the node The third index is enumerated below (see Ind3_nodes).

- `samples` stores the set of sample indexes (i.e. ids) associated with each `treeId`. The second index represents the `treeId`. The third index represents the sample number. The value is the id of the sample in the original training dataset. `{samples, treeId, sampleNum} -> origId`.
- `classWeights` (`ClassificationForest` only) stores the weights associated with each class label. The second index represents the class label. The value is the weight. `{classWeights, classLabel} -> weight`. Class weights are only stored for `Classification Forests`.

Children

1. `Ind1` : Index 1 represents the category of data within the model
2. `Ind3_Nodes` : For tree node data (i.e

IND1 Ind1

`LT_Types \ Forest_Model \`

	Ind1
--	-------------

Index 1 represents the category of data within the model

VALUE `reserved` = 1. Reserved for future use.

VALUE `nodes` = 2. The set of tree nodes within the model.

VALUE `samples` = 3. The particular record ids that are included in tree's sample .

VALUE `classWeights` = 4. The weights assigned to each class (for `ClassificationForest` only).

Children

1. `reserved` : No Documentation Found
2. `nodes` : No Documentation Found
3. `samples` : No Documentation Found
4. `classWeights` : No Documentation Found

RESERVED reserved

[LT_Types](#) \ [Forest_Model](#) \ [Ind1](#) \

t_index	reserved
-------------------------	----------

No Documentation Found

RETURN UNSIGNED4 —

NODES nodes

[LT_Types](#) \ [Forest_Model](#) \ [Ind1](#) \

t_index	nodes
-------------------------	-------

No Documentation Found

RETURN UNSIGNED4 —

SAMPLES samples

[LT_Types](#) \ [Forest_Model](#) \ [Ind1](#) \

t_index	samples
-------------------------	---------

No Documentation Found

RETURN UNSIGNED4 —

CLASSWEIGHTS `classWeights`

[LT_Types](#) \ [Forest_Model](#) \ [Ind1](#) \

<code>t_index</code>	<code>classWeights</code>
----------------------	---------------------------

No Documentation Found

RETURN UNSIGNED4 —

IND3_NODES `Ind3_Nodes`

[LT_Types](#) \ [Forest_Model](#) \

<code>Ind3_Nodes</code>

For tree node data (i.e. `Ind1 = nodes`), the following constant definitions are used for the different fields of the tree-node. Note that `Ind1` indicates tree nodes, `Ind2` represents the different nodes and `Ind3` defines the different fields. For example, the `treeId` for the first node would be stored at `[2,1,1]`. These correspond to the persisted fields of `TreeNodeDat` with similar names.

VALUE `treeID` = 1. The tree identifier.

VALUE `level` = 2. The level of the node within the tree.

VALUE `nodeId` = 3. The `nodeId` of this node within the tree.

VALUE `parentId` = 4. The parent node's `nodeId`.

VALUE `isLeft` = 5. Left / Right indicator of this node within it's parent's children.

VALUE `number` = 6. The field number to split on.

VALUE `value` = 7. The value to compare against.

VALUE `isOrd` = 8. Indicator of ordered vs categorical data.

VALUE `depend` = 9. The value to predict for samples in this leaf.

VALUE `support` = 10. The number of datapoints from the training data that reached this node.

VALUE `if` = 11. The 'impurity reduction' achieved by this branch.

Children

1. [treeId](#) : No Documentation Found
 2. [level](#) : No Documentation Found
 3. [nodeId](#) : No Documentation Found
 4. [parentId](#) : No Documentation Found
 5. [isLeft](#) : No Documentation Found
 6. [number](#) : No Documentation Found
 7. [value](#) : No Documentation Found
 8. [isOrd](#) : No Documentation Found
 9. [depend](#) : No Documentation Found
 10. [support](#) : No Documentation Found
 11. [ir](#) : No Documentation Found
-

TREEID treeId

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	treeId
-------------------------	--------

No Documentation Found

RETURN UNSIGNED4 —

LEVEL level

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	level
-------------------------	-------

No Documentation Found

RETURN UNSIGNED4 —

NODEID nodeId

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	nodeId
-------------------------	--------

No Documentation Found

RETURN UNSIGNED4 —

PARENTID parentId

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	parentId
-------------------------	----------

No Documentation Found

RETURN UNSIGNED4 —

ISLEFT isLeft

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	isLeft
-------------------------	--------

No Documentation Found

RETURN UNSIGNED4 —

NUMBER number

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	number
---------	--------

No Documentation Found

RETURN UNSIGNED4 —

VALUE value

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	value
---------	-------

No Documentation Found

RETURN UNSIGNED4 —

ISORD isOrd

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	isOrd
---------	-------

No Documentation Found

RETURN UNSIGNED4 —

DEPEND depend

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	depend
-------------------------	--------

No Documentation Found

RETURN UNSIGNED4 —

SUPPORT support

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	support
-------------------------	---------

No Documentation Found

RETURN UNSIGNED4 —

IR ir

[LT_Types](#) \ [Forest_Model](#) \ [Ind3_Nodes](#) \

t_index	ir
-------------------------	----

No Documentation Found

BF_MODEL Bf_Model

LT_Types \

	Bf_Model
--	-----------------

Definition of the meaning of the indexes of the Gradient Boosting Model variables.

Ind1 enumerates the first index, which is used to determine which type of data is stored:

- fModels stores the list of forest models that comprise the boosting hierarchy. Each of these models can be decomposed by the Forest learning modules.
- Other values are reserved for future use.

Children

1. [Ind1](#) : Index 1 represents the category of data within the model

IND1 Ind1

LT_Types \ Bf_Model \

	Ind1
--	-------------

Index 1 represents the category of data within the model

VALUE reserved = 1. Reserved for future use.

VALUE fModels = 2. The set of forest models that comprise the boosting hierarchy.

Children

1. [reserved](#) : No Documentation Found
2. [fModels](#) : No Documentation Found

RESERVED reserved

[LT_Types](#) \ [Bf_Model](#) \ [Ind1](#) \

t_index	reserved
-------------------------	----------

No Documentation Found

RETURN UNSIGNED4 —

FMODELS fModels

[LT_Types](#) \ [Bf_Model](#) \ [Ind1](#) \

t_index	fModels
-------------------------	---------

No Documentation Found

RETURN UNSIGNED4 —

GENFIELD GenField

[LT_Types](#) \

	GenField
--	----------

GenField extends NumericField by adding an isOrdinal field. This allows both Ordered and Nominal (Categorical) data to be held by the same record type.

FIELD wi ||| UNSIGNED2 — The work-item identifier for this cell.

FIELD id ||| UNSIGNED8 — The record-identifier for this cell.

FIELD number ||| UNSIGNED4 — The field number (i.e. featureId) of this cell.

FIELD value ||| REAL8 — The numerical value of this cell.

FIELD isOrdinal ||| BOOLEAN — TRUE if this field represents ordered data. FALSE if it is categorical.

SEE ML_Core.Types.NumericField.

TREENODEDAT **TreeNodeDat**

[LT_Types](#) \

TreeNodeDat

This is the major working structure for building the forest.

For efficiency and uniformity, this record structure serves several purposes as the forest is built:

- It represents all of the X,Y data associated with each tree and node as the forest is being built. This case is recognized by `id > 0` (i.e. it is a data point). `wi`, `treeId`, `level`, and `NodeId` represent the work-item and tree node with which the data is currently associated. All data in a tree's sample is originally assigned to the tree's root node (`level = 1`, `nodeId = 1`).
 - `id` is the sample index in this trees data bootstrap sample.
 - `origId` is the sample index in the original Independent(X) data.
 - `number` is the field number from the X data.
 - `isOrdinal` indicates whether this data is Ordinal (true) or Nominal (false).
 - `value` is the data value of this data point.
 - `depend` is the Dependent (Y) value associated with this data point.
- It represents the skeleton of the tree as the tree is built from the root down and the data points are subsumed (summarized) by the evolving tree structure. These cases can be identified by `id = 0`.
 - It represents branch (split) nodes:
 - * `id = 0` – All data was subsumed.
 - * `number > 0` – The original field number of the Independent(X) variable on which to split.
 - * `value` – the value on which to split

- * `parentId` – The `nodeId` of the branch at the previous level that leads to this node. Zero only for root.
- * `level` – The distance from the root (`root = 1`).
- * `support` – The number of data points that reach this node.
- * `ir` – The impurity reduction for this split.
- It represents leaf nodes:
 - * `id = 0` – All data was subsumed.
 - * `number = 0` – This discriminates a leaf from a branch node.
 - * `depend` has the Y value for that leaf.
 - * `parentId` has the `nodeId` of the branch node at the previous level.
 - * `support` has the count of samples that reached this leaf.
 - * `level` – The depth of the node in the tree (`root = 1`).

Each tree starts with all sampled data points assigned to the root node (i.e. `level = 1`, `nodeId = 1`). As the trees grow, data points are assigned to deeper branches, and eventually to leaf nodes, where they are ultimately subsumed (summarized) and removed from the dataset.

At the end of the forest growing process only the tree skeleton remains – all the datapoints having been summarized by the resulting branch and leaf nodes.

FIELD `treeId` ||| UNSIGNED4 — The unique id of the tree in the forest.

FIELD `nodeId` ||| UNSIGNED8 — The id of this node within the tree.

FIELD `parentId` ||| UNSIGNED8 — The node id of this node’s parent.

FIELD `isLeft` ||| BOOLEAN — Indicates whether this node is the left child or the right child of the parent.

FIELD `wi` ||| UNSIGNED2 — The work item with which this record is associated.

FIELD `id` ||| UNSIGNED8 — The record id of the sample during tree construction. Will be zero once the record has been replaced by a skeleton node (i.e. branch or leaf).

FIELD `number` ||| UNSIGNED4 — The field number on which the branch splits

FIELD `value` ||| REAL8 — The value of the data field, or the `splitValue` for a branch node.

FIELD `level` ||| UNSIGNED2 — The level of the node within its tree. Root is 1.

FIELD `origId` ||| INTEGER4 — The sample index (`id`) of the original X data that this sample came from.

FIELD `depend` ||| REAL8 — The dependent value associated with this id.

FIELD `support` ||| UNSIGNED8 — The number of data samples subsumed by this node.

FIELD `ir` ||| REAL8 — The ‘impurity’ reduction achieved by this branch.

FIELD `observWeight` ||| REAL8 — The observation weight associated with this observation.

FIELD `isordinal` ||| BOOLEAN — No Doc

BFTREENODEDAT BfTreeNodeDat

[LT_Types](#) \

BfTreeNodeDat

Main data structure for processing Boosted Forest.

The structure is the same as for random forests, but with an extra field gbLevel that represents the level of the gradient boosted forest nodes within the boosting hierarchy.

Each set of nodes representing a forest is organized hierarchically based on that field.

Each level of the Boosted Forest contains a random forest. The results from each random forest are added together to get the final result for the GBF.

FIELD treeid ||| UNSIGNED4 — No Doc

FIELD nodeid ||| UNSIGNED8 — No Doc

FIELD parentid ||| UNSIGNED8 — No Doc

FIELD isleft ||| BOOLEAN — No Doc

FIELD wi ||| UNSIGNED2 — No Doc

FIELD id ||| UNSIGNED8 — No Doc

FIELD number ||| UNSIGNED4 — No Doc

FIELD value ||| REAL8 — No Doc

FIELD isordinal ||| BOOLEAN — No Doc

FIELD level ||| UNSIGNED2 — No Doc

FIELD origid ||| INTEGER4 — No Doc

FIELD depend ||| REAL8 — No Doc

FIELD support ||| UNSIGNED8 — No Doc

FIELD ir ||| REAL8 — No Doc

FIELD observweight ||| REAL8 — No Doc

FIELD bflevel ||| UNSIGNED2 — No Doc

CLASSPROBS ClassProbs

[LT_Types](#) \

	ClassProbs
--	------------

The probability that a given sample is of a given class

FIELD wi ||| UNSIGNED2 — The work-item identifier.

FIELD id ||| UNSIGNED8 — The record-id of the sample.

FIELD class ||| INTEGER4 — The class label.

FIELD cnt ||| INTEGER4 — The number of trees that predicted this class label.

FIELD prob ||| REAL8 — The percentage of trees that assigned this class label, which is a rough stand-in for the probability that the label is correct.

NODESUMMARY NodeSummary

[LT_Types](#) \

	NodeSummary
--	-------------

NodeSummary provides information to identify a given tree node

FIELD wi ||| UNSIGNED2 — The work-item id for this node.

FIELD treeId ||| UNSIGNED4 — The tree identifier within this work-item.

FIELD nodeId ||| UNSIGNED8 — The node within the tree and work-item.

FIELD parentId ||| UNSIGNED8 — The nodeId of this nodes parent node.

FIELD isLeft ||| BOOLEAN — Boolean indicator of whether this is the Left child (TRUE) or Right child (FALSE) of the parent.

FIELD support ||| UNSIGNED8 — The number of data samples that reached this node.

SPLITDAT SplitDat

[LT_Types](#) \

	SplitDat
--	----------

SplitDat is used to hold information about a potential split. It is based on the NodeSummary record type above. It adds the following fields

FIELD number ||| UNSIGNED4 — The field number of the Independent data that is being used to split.

FIELD splitVal ||| REAL8 — The value by which to split the data.

FIELD isOrdinal ||| BOOLEAN — TRUE indicates that it is an ordered value and will use a greater-than-or-equal split (i.e. value >= splitVal). FALSE indicates that the values are nominal (i.e. categorical) and will use an equal-to split (i.e. value = splitVal)

FIELD wi ||| UNSIGNED2 — No Doc

FIELD treeid ||| UNSIGNED4 — No Doc

FIELD nodeid ||| UNSIGNED8 — No Doc

FIELD parentid ||| UNSIGNED8 — No Doc

FIELD isleft ||| BOOLEAN — No Doc

FIELD support ||| UNSIGNED8 — No Doc

FIELD ir ||| REAL8 — No Doc

NODEIMPURITY NodeImpurity

[LT_Types](#) \

	NodeImpurity
--	--------------

NodeImpurity carries identifying information for a node as well as its impurity level. It is based on the NodeSummary record type above, but includes an assessment of the 'impurity' of the data at this node (i.e. GINI, Variance, Entropy).

FIELD impurity ||| REAL8 — The level of impurity at the given node. Zero is most pure.

- FIELD** wi ||| UNSIGNED2 — No Doc
- FIELD** treeid ||| UNSIGNED4 — No Doc
- FIELD** nodeid ||| UNSIGNED8 — No Doc
- FIELD** parentid ||| UNSIGNED8 — No Doc
- FIELD** isleft ||| BOOLEAN — No Doc
- FIELD** support ||| UNSIGNED8 — No Doc

WIINFO wiInfo

[LT_Types](#) \

	wiInfo
--	---------------

Provides a summary of each work item for use in building the forest.

- FIELD** wi ||| UNSIGNED2 — The work-item identifier.
- FIELD** numSamples ||| UNSIGNED8 — The number of samples within this work-item
- FIELD** numFeatures ||| UNSIGNED4 — The number of features (i.e. number fields in the Independent data for this work-item.
- FIELD** featuresPerNode ||| UNSIGNED8 — The number of features to be randomly chosen at each level of tree building. It is a function of, the user parameter 'featuresPerNode' and the number of features in the work-item numFeatures.

MODELSTATS ModelStats

[LT_Types](#) \

	ModelStats
--	-------------------

Model Statistics Record Provides descriptive information about a Model

FIELD	<u>wi</u> UNSIGNED2	— The work-item whose model is described
FIELD	<u>treeCount</u> UNSIGNED8	— The number of trees in the forest
FIELD	<u>minTreeDepth</u> UNSIGNED8	— The depth of the shallowest tree
FIELD	<u>maxTreeDepth</u> UNSIGNED8	— The depth of the deepest tree
FIELD	<u>avgTreeDepth</u> REAL8	— The average depth of all trees
FIELD	<u>minTreeNodes</u> UNSIGNED8	— The number of nodes in the smallest tree
FIELD	<u>maxTreeNodes</u> UNSIGNED8	— The number of nodes in the biggest tree
FIELD	<u>avgTreeNodes</u> REAL8	— The average number of nodes for all trees
FIELD	<u>totalNodes</u> UNSIGNED8	— The number of nodes in the forest
FIELD	<u>minSupport</u> UNSIGNED8	— The minimum sum of support for all trees. Support indicates the number of training datapoints that arrived at a given leaf node
FIELD	<u>maxSupport</u> UNSIGNED8	— The maximum sum of support for all trees
FIELD	<u>agvSupport</u>	— The average sum of support for all trees
FIELD	<u>avgSupportPerLeaf</u> REAL8	— The average number of data points per leaf across the forest
FIELD	<u>maxSupportPerLeaf</u> UNSIGNED8	— The maximum data points at any single leaf across the forest
FIELD	<u>avgLeafDepth</u> REAL8	— The average depth for all leaf nodes for all trees
FIELD	<u>minLeafDepth</u> UNSIGNED8	— The minimum depth for all leaf nodes for all trees
FIELD	<u>avgsupport</u> REAL8	— No Doc
FIELD	<u>bflvel</u> UNSIGNED8	— No Doc

FEATUREIMPORTANCEREC FeatureImportanceRec

[LT_Types](#) \

	FeatureImportanceRec
--	----------------------

Feature Importance Record describes the importance of each feature.

FIELD	<u>wi</u> UNSIGNED2	— The work-item associated with this information.
--------------	-------------------------	---

FIELD number ||| UNSIGNED4 — The feature number.

FIELD importance ||| REAL8 — The 'importance' metric. Higher value is more important.

FIELD uses ||| UNSIGNED8 — The number of times the feature was used in the forest.

CLASSWEIGHTSREC ClassWeightsRec

[LT_Types](#) \

	ClassWeightsRec
--	-----------------

ClassWeightsRecord holds the weights associated with each class label.

FIELD wi ||| UNSIGNED2 — The work-item.

FIELD classLabel ||| INTEGER4 — The subject class label.

FIELD weight ||| REAL8 — The weight associated with this class label.

LUCI_SCORECARD LUCI_Scorecard

[LT_Types](#) \

	LUCI_Scorecard
--	----------------

Structure used to describe the Scorecards for LUCI format export. For a single scorecard model, a single LUCI_Scorecard record is used. For multiple scorecards, one record is required per scorecard. One L2SC or L2FO record will be generated per scorecard, and additionally One L2SE record will be generated for each scorecard with a non-blank 'filter_expr'.

FIELD wi_num ||| UNSIGNED8 — The work-item number on which to base this scorecard or '1' if only one work-item / scorecard is used.

FIELD scorecard_name ||| STRING — The LUCI name for this scorecard.

FIELD filter_expr ||| STRING — Optional – An expression on the LUCI input dataset layout that selects the records to be included in this scorecard (e.g. 'state_id = 2'). If the expression contains strings, the single-quotes must be preceded by a backslash escape character (e.g. 'state = \'NY\"]'). The filter expression must follow ECL Boolean expression syntax. It should be blank if all records are to be used. See L2SE LUCI record format, Scorecard-Election-Criteria for more details.

FIELD fieldMap ||| TABLE (Field_Mapping) — A DATASET(Field_Mapping) as returned from the FromField macro that maps the Field Names (as used in the LUCI definition) to the field numbers (as used in the ML model). Note: must be the same set of fields used in training the forest for this work item.

LUCI_Export

[Go Up](#)

IMPORTS

LT__Types | std.Str | std.system.ThorLib | __versions.ML_Core.V3_2_2.ML_Core |
__versions.ML_Core.V3_2_2.ML_Core.Types |
__versions.ML_Core.V3_2_2.ML_Core.ModelOps2 |

DESCRIPTIONS

LUCI_EXPORT LUCI_Export

<code>/ EXPORT DATASET(LUCI_Rec)</code>	LUCI_Export
<code>(DATASET(Layout_Model2) mod, STRING model_id, STRING model_name, DATASET(LUCI_Scorecard) scorecards)</code>	

Export a Learning Forest model to LUCI format.

LUCI is a LexisNexis proprietary mechanism for describing a model that can then be efficiently processed within an LN product.

Note the following:

- This module produces a LUCI file that outputs the exact same Regression values as the Regression Forest model. LUCI allows some additional features that are beyond the scope of this module. If these features are needed, the resultant LUCI file may need to be hand edited to achieve those results. Examples of these features include:
 - Defining Reason Code logic (L1MD record)
 - Setting minimum and maximum bounds (L2FO record)

- Adding an increment value to the final results (L2FO)
- Setting a scaling formula to scale the final results (L2FO)
- Excluding certain input records (L1EX record)
- See the LUCI documentation for more info:
https://gitlab.ins.risk.regn.net/HIPIE/HIPIE_Plugins/wikis/LUCIfiles
- This module supports the following LUCI use cases:
 - Single work-item / single scorecard.
 - Work-items and corresponding scorecards represent training of different response variables on (potentially) different subsets of the features in the LUCI input layout.
 - Work-items and corresponding scorecards represent training of the same response variable across subsets of the input data (e.g. one per country). It is anticipated, though not required, that the same subset of LUCI input layout features was used for training each subset.

The following types of LUCI record are created:

- A single L1MD record.
- One L2FO record per LUCI scorecard.
- One L2SE record per scorecard that includes a filter expression.
- One L3TN record per node for each tree in each forest (i.e. work-item).

Note that scorecards in LUCI correspond to work-items in LearningForest.

PARAMETER mod ||| TABLE (Layout_Model2) — The random forest model as returned from GetModel.

PARAMETER model_name ||| STRING — The name of the LUCI model (see LUCI L1MD definition).

PARAMETER model_id ||| STRING — The id of the LUCI model (see LUCI L1MD definition).

PARAMETER scorecards ||| TABLE (LUCI_Scorecard) — DATASET(LUCI_Scorecard) describing each work-item in the model that will be exported as a LUCI scorecard.

RETURN TABLE (LUCI_Rec) — DATASET(LUCI_Rec) representing the lines of a LUCI .csv file. The caller is responsible for melding the lines into an actual .csv file and storing it in a given location.

SEE LT_Types.LUCI_Scorecard, LT_Types.LUCI_Rec

RegressionForest

[Go Up](#)

IMPORTS

```
__versions.ML_Core.V3_2_2.ML_Core |  
__versions.ML_Core.V3_2_2.ML_Core.Types | ml_core.interfaces | LT_Types |  
internal |
```

DESCRIPTIONS

REGRESSIONFOREST RegressionForest

	RegressionForest
<code>(UNSIGNED numTrees=100, UNSIGNED featuresPerNode=0, UNSIGNED maxDepth=100, SET OF UNSIGNED nominalFields=[])</code>	

Regression using Random Forest algorithm. This module implements Random Forest regression as described by Breiman, 2001 with extensions (see <https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf>).

Random Forests provide an effective method for regression. They are known to be one of the best out-of-the-box methods as there are few assumptions made regarding the nature of the data or its relationships. Random Forests can effectively manage large numbers of features, and will automatically choose the most relevant features.

Regression Forests can handle non-linear and discontinuous relationships among features.

One limitation of Regression Forests is that they provide no extrapolation beyond the bounds of the training data. The training set should extend to the limits of expected feature values.

This implementation allows both Ordinal (discrete or continuous) and Nominal (unordered categorical

values) for the independent (X) features. There is therefore, no need to one-hot encode categorical features. Nominal features should be identified by including their feature 'number' in the set of 'nominalFields'.

RegressionForest supports the Myriad interface meaning that multiple independent models can be computed with a single call (see `ML_Core.Types` for information on using the Myriad feature).

Notes on use of NumericField layouts:

- Work-item ids ('wi' field) are not required to be sequential, though they must be positive numbers. It is a good practice to assign $wi = 1$ when only one work-item is used.
- Record Ids ('id' field) are not required to be sequential, though slightly faster performance will result if they are sequential (i.e. $1 \dots numRecords$) for each work-item.
- Feature numbers ('number' field) are not required to be sequential, though slightly faster performance will result if they are (i.e. $1 \dots numFeatures$) for each work-item.

PARAMETER numTrees ||| UNSIGNED8 — The number of trees to create as the forest for each work-item. This defaults to 100, which is adequate for most cases. Increasing this parameter generally results in less variance in accuracy between runs, at the expense of greater run time.

PARAMETER featuresPerNode ||| UNSIGNED8 — The number of features to choose among at each split in each tree. This number of features will be chosen at random from the full set of features. The default value (0) uses the square root of the number of features provided, which works well for most cases.

PARAMETER maxDepth ||| UNSIGNED8 — The deepest to grow any tree in the forest. The default is 100, which is adequate for most purposes. Increasing this value for very large and complex problems may provide slightly greater accuracy at the expense of much greater runtime.

PARAMETER nominalFields ||| SET (UNSIGNED8) — An optional set of field 'numbers' that represent Nominal (i.e. unordered, categorical) values. Specifying the nominal fields improves run-time performance on these fields and may improve accuracy as well. Binary fields (fields with only two values) need not be included here as they can be considered either ordinal or nominal. The default is to treat all fields as ordered. Note that this feature should only be used if all of the independent data for all work-items use the same record format, and therefore have the same set of nominal fields.

PARENT LearningForest

PARENT iregression2

Children

1. [GetModelStats](#) : Get summary statistical information about the model
2. [Model2Nodes](#) : Extract the set of tree nodes from a model

3. [Accuracy](#) : Assess the accuracy of a set of predictions
4. [FeatureImportance](#) :
Determine the relative importance of features in the decision process of the model
5. [GetModel](#) : Fit a model that maps independent data (X) to its class (Y)
6. [Predict](#) : Predict a set of data points using a previously fitted model
7. [DecisionDistanceMatrix](#) :
Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS)
8. [UniquenessFactor](#) : Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points
9. [CompressModel](#) : Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child

GETMODELSTATS `GetModelStats`

[RegressionForest](#) \

DATASET (<code>ModelStats</code>)	<code>GetModelStats</code>
<code>(DATASET(Layout_Model2) mod)</code>	

Get summary statistical information about the model.

PARAMETER `mod` ||| TABLE (Layout_Model2) — A model previously returned from `GetModel`.

RETURN TABLE ({ UNSIGNED2 `wi` , UNSIGNED8 `treeCount` , UNSIGNED8 `minTreeDepth` , UNSIGNED8 `maxTreeDepth` , REAL8 `avgTreeDepth` , UNSIGNED8 `minTreeNodees` , UNSIGNED8 `maxTreeNodees` , REAL8 `avgTreeNodees` , UNSIGNED8 `totalNodes` , UNSIGNED8 `minSupport` , UNSIGNED8 `maxSupport` , REAL8 `avgSupport` , REAL8 `avgSupportPerLeaf` , UNSIGNED8 `maxSupportPerLeaf` , REAL8 `avgLeafDepth` , UNSIGNED8 `minLeafDepth` , UNSIGNED8 `bfLevel` }) — A single `ModelStats` record per work-item, containing information about the model for that work-item.

SEE `LT_Types.ModelStats`

INHERITED

MODEL2NODES Model2Nodes

[RegressionForest](#) \

DATASET(TreeNodeDat)	Model2Nodes
(DATASET(Layout_Model2) mod)	

Extract the set of tree nodes from a model.

PARAMETER mod ||| TABLE (Layout_Model2) — A model as returned from GetModel.

RETURN TABLE ({ UNSIGNED4 treeId , UNSIGNED8 nodeId , UNSIGNED8 parentId , BOOLEAN isLeft , UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value , BOOLEAN isOrdinal , UNSIGNED2 level , INTEGER4 origId , REAL8 depend , UNSIGNED8 support , REAL8 ir , REAL8 observWeight }) — Set of tree nodes representing the fitted forest in DATASET(TreeNodeDat) format.

SEE LT_Types.TreeNodeDat

INHERITED

ACCURACY Accuracy

[RegressionForest](#) \

DATASET(Regression_Accuracy)	Accuracy
(DATASET(Layout_Model2) model, DATASET(NumericField) actuals, DATASET(NumericField) observations)	

Assess the accuracy of a set of predictions. This is equivalent to calling predict and then Analysis.Regression.Accuracy.

PARAMETER model ||| TABLE (Layout_Model2) — The model as returned from GetModel

PARAMETER actuals ||| TABLE (NumericField) — The actual values of the dependent variable to compare with the predictions.

PARAMETER observations ||| TABLE (NumericField) — The independent data upon which the accuracy assessment is to be based.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 regressor , REAL8 R2 , REAL8 MSE , REAL8 RMSE }) — Accuracy statistics (see Types.Regression_Accuracy for details)

OVERLOAD

FEATUREIMPORTANCE FeatureImportance

RegressionForest \

	FeatureImportance
	(DATASET(Layout_Model2) mod)

Determine the relative importance of features in the decision process of the model. Calculate feature importance using the Mean Decrease Impurity (MDI) method from "Understanding Random Forests: by Gilles Loupe (<https://arxiv.org/pdf/1407.7502.pdf>) and due to Breiman [2001, 2002].

Each feature is ranked by:

```
SUM for each branch node in which feature appears (across all trees):  
(impurity\_reduction * number of nodes split) / numTrees.
```

PARAMETER mod ||| TABLE (Layout_Model2) — The model to use for ranking of feature importance.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED4 number , REAL8 importance , UNSIGNED8 uses }) — DATASET(FEATUREIMPORTANCEREC), one per feature per wi.

SEE LT_Types.FeatureImportanceRec

INHERITED

GETMODEL GetModel

RegressionForest \

GetModel
(DATASET(NumericField) <i>independents</i> , DATASET(NumericField) <i>dependents</i>)

Fit a model that maps independent data (X) to its class (Y).

PARAMETER *independents* ||| TABLE (NumericField) — The set of independent data in NumericField format.

PARAMETER *dependents* ||| TABLE (NumericField) — The dependent variable in NumericField format. The 'number' field is not used as only one dependent variable is currently supported. For consistency, it should be set to 1.

RETURN TABLE ({ UNSIGNED2 *wi* , REAL8 *value* , SET (UNSIGNED4) *indexes* })
— Model in Layout_Model2 format describing the fitted forest.

SEE ML_Core.Types.NumericField, ML_Core.Types.Layout_Model2

OVERRIDE

PREDICT Predict

RegressionForest \

DATASET(NumericField)	Predict
(DATASET(Layout_Model2) <i>model</i> , DATASET(NumericField) <i>observations</i>)	

Predict a set of data points using a previously fitted model.

PARAMETER *mod* ||| — A model previously returned by GetModel in Layout_Model2 format.

PARAMETER *observations* ||| TABLE (NumericField) — The set of independent data in NumericField format.

PARAMETER *model* ||| TABLE (Layout_Model2) — No Doc

RETURN TABLE ({ UNSIGNED2 *wi* , UNSIGNED8 *id* , UNSIGNED4 *number* , REAL8 *value* }) — A NumericField dataset that provides a prediction for each record in observations.

OVERRIDE

DECISIONDISTANCEMATRIX DecisionDistanceMatrix

RegressionForest \

DecisionDistanceMatrix
(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)

Calculate a matrix of distances between data points in Random Forest Decision Space (RFDS). This is an experimental method and may not scale to large numbers of data point combinations. Two sets of data points X1 and X2 are taken as parameters. A Decision Distance will be returned for every point in X1 to every point in X2. Therefore, if X1 has N points and X2 has M points, an N x M matrix of results will be produced. X2 may be omitted, in which case, an N x N matrix will be produced with a Decision Distance for every pair of points in X1.

This metric represents a distance measure in the RFDS. As such, it provides a continuous measure of distance in a space that is highly non-linear and discontinuous relative to the training data. Distances in RFDS can be thought of as the number of binary decisions that separate two points in the tree. DD, however is a normalized metric $0 \leq DD < 1$ that incorporates the depth of the decision tree. It is also averaged over all of the trees in the forest. It can possibly be viewed as an approximation of the relative Hamming Distances between points.

PARAMETER mod ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER X1 ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER X2 ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'number' is the id of the "to" point. 'value' contains the DD metric between "from" and "to" points. Note that if the same point is in X1 and X2, there will be redundant metrics, since DD is a symmetric measure (i.e. $DD(x1, x2) = DD(x2, x1)$).

INHERITED

UNIQUENESSFACTOR UniquenessFactor

[RegressionForest](#) \

UniquenessFactor
(DATASET(Layout_Model2) mod, DATASET(NumericField) X1, DATASET(NumericField) X2=empty_data)

Uniqueness Factor is an experimental metric that determines how far a given point is (in Random Forest Decision Distance) from a set of other points. It may not scale to large numbers of data points. Uniqueness Factor looks at the Decision Distance from each point to every other point in a set. It is similar to Decision Distance (above), but rather than providing a distance of each "from" point to every "to" point, it provides the average distance of each "from" point to all of the "to" points. Like Decision Distance, UF lies on the interval: $0 \leq UF < 1$. A high value of UF may indicate an anomolous data point, while a low value may indicate "typicalness" of a data point. It may therefore have utility for anomaly detection or conversely, for the identification of class prototypes (e.g. the members of a class with the lowest UF). In a two-step process one could potentially compute class prototypes and then look at the distance of a point from all class prototypes. This could result in a way to detect anomalies with respect to e.g., known usage patterns.

PARAMETER mod ||| TABLE (Layout_Model2) — The Random Forest model on which to base the distances.

PARAMETER X1 ||| TABLE (NumericField) — DATASET(NumericField) of "from" points.

PARAMETER X2 ||| TABLE (NumericField) — (Optional) DATASET(NumericField) of "to" points. If this parameter is omitted, the X1 will be used as both "to" and "from" points.

RETURN TABLE ({ UNSIGNED2 wi , UNSIGNED8 id , UNSIGNED4 number , REAL8 value }) — DATASET(NumericField) matrix where 'id' is the id of the "from" point and 'value' contains the UF metric for the point. I.e. the average DD of the "from" point to all "to" points. The 'number' field is not used and is set to 1.

INHERITED

COMPRESSMODEL CompressModel

[RegressionForest](#) \

CompressModel
(DATASET(Layout_Model2) mod)

Compress and cleanup the model This function is provided to reduce the size of a model by compressing out branches with only one child. These branches are a result of the RF algorithm, and do not affect the results of the model. This is an expensive operation, which is why it is not done as a matter of course. It reduces the size of the model somewhat, and therefore slightly speeds up any processing that uses the model, and reduces storage size. You may want to compress the model if storage is at a premium, or if the model is to be used many times (so that the slight performance gain is multiplied). This also makes the model somewhat more readable, and could be useful when analyzing the tree or converting it to another system (e.g. LUCI) for processing.

PARAMETER **mod** ||| TABLE (Layout_Model2) — Model as returned from GetModel in Layout_Model2 format.

RETURN TABLE ({ UNSIGNED2 wi , REAL8 value , SET (UNSIGNED4) indexes })
— The Compressed Model.

SEE ML_Core.Types.Layout_Model2

INHERITED
