Package 'MicrosoftML'

February 11, 2018

Version 9.3.0

MmlBuildId 9.3.0.1210

Title Microsoft Machine Learning for R

Author Microsoft Corporation

Maintainer Microsoft Corporation <mrspack@microsoft.com>

Depends R (>= 3.3.2),
methods,
RevoScaleR (>= 9.2.1)

Imports stats,
utils,
tools,
jsonlite

Suggests RUnit,
lattice,
magrittr,
knitr

#VignetteBuilder knitr

Description Microsoft Machine Learning algorithms for R.

License file LICENSE

Copyright Copyright 2016 Microsoft Corporation

RoxygenNote 5.0.1

R topics documented:

`MicrosoftML-package` *MicrosoftML: State-of-the-Art Machine Learning Algorithms from Microsoft Corporation*

Description

A package that provides state-of-the-art machine learning algorithms for R, developed by Microsoft. It is used with the **RevoScaleR** package.

Details

Package:	MicrosoftML
Type:	Package
Version:	9.3.0
Build:	9.3.0.1210
License:	file LICENSE
LazyLoad:	yes

The key functions/concepts in the package are as follows (to list all public functions, type library(help="MicrosoftML") at the R prompt):

Machine Learning Algorithms
- `rxFastTrees`: An implementation of FastRank, an efficient implementation of the MART gradient boosting algorithm.
- `rxFastForest`: A random forest and Quantile regression forest implementation using `rxFastTrees`.
- `rxLogisticRegression`: Logistic regression using L-BFGS.
- `rxOneClassSvm`: One class support vector machines.
- `rxNeuralNet`: Binary, multi-class, and regression neural net.
- `rxFastLinear`: Stochastic dual coordinate ascent optimization for linear binary classification and regression.

Scoring
- `rxPredict.mlModel`: Scores using a model created by one of the machine learning algorithms.

Helper functions for arguments
- `expLoss`: Specifications for exponential classification loss function.
- `logLoss`: Specifications for log classification loss function.
- `hingeLoss`: Specifications for hinge classification loss function.
- `smoothHingeLoss`: Specifications for smooth hinge classification loss function.
- `poissonLoss`: Specifications for poisson regression loss function.
- `squaredLoss`: Specifications for squared regression loss function.
- `linearKernel`: Specification for linear kernel.
- `rbfKernel`: Specification for radial basis function kernel.
- `polynomialKernel`: Specification for polynomial kernel.
- `sigmoidKernel`: Specification for sigmoid kernel.
- `minCount`: Specification for feature selection in count mode.
- `mutualInformation`: Specification for feature selection in mutual information mode.

Helper functions for machine learning transforms

- `featurizeText`: Transformation to produce a bag of counts of ngrams in a given text. It offers language detection, tokenization, stopwords removing, text normalization and feature generation.
- `concat`: Transformation to create a single vector-valued column from multiple columns.
- `categorical`: Create indicator vector using categorical transform with dictionary.
- `categoricalHash`: Converts the categorical value into an indicator array by hashing.
- `selectFeatures`: Selects features from the specified variables.

Author(s)

Microsoft Corporation Microsoft Technical Support

`categorical` *Machine Learning Categorical Data Transform*

Description

Categorical transform that can be performed on data before training a model.

Usage

```
categorical(vars, outputKind = "ind", maxNumTerms = 1e+06, terms = "",
  ...)
```

Arguments

`vars`	A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.
`outputKind`	A character string that specifies the kind of output kind. • `"ind"`: Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column. • `"bag"`: Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent • `"key"`: Outputs an index. The output is an integer id (between 1 and the number of categories in the dictionary) of the category. The default value is `"ind"`.
`maxNumTerms`	An integer that specifies the maximum number of categories to include in the dictionary. The default value is 1000000.
`terms`	Optional character vector of terms or categories.
`...`	Additional arguments sent to compute engine.

Details

The `categorical` transform passes through a data set, operating on text columns, to build a dictionary of categories. For each row, the entire text string appearing in the input column is defined as a category. The output of the categorical transform is an indicator vector. Each slot in this vector corresponds to a category in the dictionary, so its length is the size of the built dictionary. The categorical transform can be applied to one or more columns, in which case it builds a separate dictionary for each column that it is applied to.

`categorical` is not currently supported to handle factor data.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastTrees`, `rxFastForest`, `rxNeuralNet`, `rxOneClassSvm`, `rxLogisticRegression`.

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )
    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)


# Use a categorical transform: the entire string is treated as a category
outModel1 <- rxLogisticRegression(like~reviewCat, data = trainReviews,
    mlTransforms = list(categorical(vars = c(reviewCat = "review"))))
# Note that 'I hate it' and 'I love it' (the only strings appearing more than once)
# have non-zero weights
summary(outModel1)

# Use the model to score
scoreOutDF1 <- rxPredict(outModel1, data = testReviews,
    extraVarsToWrite = "review")
scoreOutDF1
```

`categoricalHash` *Machine Learning Categorical HashData Transform*

Description

Categorical hash transform that can be performed on data before training a model.

Usage

```
categoricalHash(vars, hashBits = 16, seed = 314489979, ordered = TRUE,
  invertHash = 0, outputKind = "Bag", ...)
```

Arguments

`vars`	A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.
`hashBits`	An integer specifying the number of bits to hash into. Must be between 1 and 30, inclusive. The default value is 16.
`seed`	An integer specifying the hashing seed. The default value is 314489979.
`ordered`	`TRUE` to include the position of each term in the hash. Otherwise, `FALSE`. The default value is `TRUE`.
`invertHash`	An integer specifying the limit on the number of keys that can be used to generate the slot name. `0` means no invert hashing; `-1` means no limit. While a zero value gives better performance, a non-zero value is needed to get meaningful coefficent names. The default value is `0`.
`outputKind`	A character string that specifies the kind of output kind.

- "ind": Outputs an indicator vector. The input column is a vector of categories, and the output contains one indicator vector per slot in the input column.
- "bag": Outputs a multi-set vector. If the input column is a vector of categories, the output contains one vector, where the value in each slot is the number of occurrences of the category in the input vector. If the input column contains a single category, the indicator vector and the bag vector are equivalent
- "key": Outputs an index. The output is an integer id (between 1 and the number of categories in the dictionary) of the category.

The default value is "Bag".

`...` Additional arguments sent to the compute engine.

Details

`categoricalHash` converts a categorical value into an indicator array by hashing the value and using the hash as an index in the bag. If the input column is a vector, a single indicator bag is returned for it.

`categoricalHash` does not currently support handling factor data.

Value

a `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastTrees`, `rxFastForest`, `rxNeuralNet`, `rxOneClassSvm`, `rxLogisticRegression`.

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )

    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)


# Use a categorical hash transform
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews,
    mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"))))
# Weights are similar to categorical
summary(outModel2)

# Use the model to score
scoreOutDF2 <- rxPredict(outModel2, data = testReviews,
    extraVarsToWrite = "review")
scoreOutDF2
```

---

concat — *Machine Learning Concat Transform*

Description

Combines several columns into a single vector-valued column.

Usage

```
concat(vars, ...)
```

Arguments

`vars`	A named list of character vectors of input variable names and the name of the output variable. Note that all the input variables must be of the same type. It is possible to produce mulitple output columns with the concatenation transform. In this case, you need to use a list of vectors to define a one-to-one mappings

	between input and output variables. For example, to concatenate columns InNameA and InNameB into column OutName1 and also columns InNameC and InNameD into column OutName2, use the list: (list(OutName1 = c(InNameA, InNameB), outName2 = c(InNameC, InNameD)))
`...`	Additional arguments sent to the compute engine

Details

`concat` creates a single vector-valued column from multiple columns. It can be performed on data before training a model. The concatenation can significantly speed up the processing of data when the number of columns is as large as hundreds to thousands.

Value

A `maml` object defining the concatenation transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`featurizeText`, `categorical`, `categoricalHash`, `rxFastTrees`, `rxFastForest`, `rxNeuralNet`, `rxOneClassSvm`, `rxLogisticRegression`.

Examples

```
testObs <- rnorm(nrow(iris)) > 0
testIris <- iris[testObs,]
trainIris <- iris[!testObs,]

multiLogitOut <- rxLogisticRegression(
        formula = Species~Features, type = "multiClass", data = trainIris,
        mlTransforms = list(concat(vars = list(
            Features = c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")
          ))))
summary(multiLogitOut)
```

`dropColumns` *Drops columns from the dataset*

Description

Specified columns to drop from the dataset.

Usage

```
dropColumns(vars, ...)
```

Arguments

`vars`	A character vector or list of the names of the variables to drop.
`...`	Additional arguments sent to compute engine.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

`ensembleControl` *ensembleControl*

Description

Control the parameters used to create an ensemble.

Usage

```
ensembleControl(randomSeed = NULL, modelCount = 1, replace = FALSE,
  sampRate = ifelse(replace, 1, 0.632), splitData = FALSE,
  combineMethod = NULL, ...)
```

Arguments

`randomSeed`	Specifies the random seed. The default value is `NULL`.
`modelCount`	Specifies the number of models to train. The default value is `1`, meaning no ensembling occurs.
`replace`	A logical value specifying if the sampling of observations should be done with or without replacement. The default value is `FALSE`.
`sampRate`	a scalar of positive value specifying the percentage of observations to sample for each trainer. The default is 1.0 for sampling with replacement (i.e., replace=TRUE) and 0.632 for sampling without replacement (i.e., replace=FALSE).
`splitData`	A logical value that specifies whether or not to train the base models on non-overlapping partitions. The default is `FALSE`. It is available only for `RxSpark` compute context and is ignored for others.
`combineMethod`	Specifies the method used to combine the models: • `median` to compute the median of the individual model outputs, • `average` to compute the average of the individual model outputs and • `vote` to compute (pos-neg) / the total number of models, where 'pos' is the number of positive outputs and 'neg' is the number of negative outputs. The default value is `median`.
`...`	Not used currently.

Value

A list of ensemble parameters.

extractPixels — *Machine Learning Extract Pixel Data Transform*

Description

Extracts the pixel values from an image.

Usage

```
extractPixels(vars, useAlpha = FALSE, useRed = TRUE, useGreen = TRUE,
  useBlue = TRUE, interleaveARGB = FALSE, convert = TRUE, offset = NULL,
  scale = NULL)
```

Arguments

`vars`	A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.
`useAlpha`	Specifies whether to use alpha channel. The default value is `FALSE`.
`useRed`	Specifies whether to use red channel. The default value is `TRUE`.
`useGreen`	Specifies whether to use green channel. The default value is `TRUE`.
`useBlue`	Specifies whether to use blue channel. The default value is `TRUE`.
`interleaveARGB`	Whether to separate each channel or interleave in ARGB order. This might be important, for example, if you are training a convolutional neural network, since this would affect the shape of the kernel, stride etc.
`convert`	Whether to convert to floating point. The default value is `FALSE`.
`offset`	Specifies the offset (pre-scale). This requires `convert = TRUE`. The default value is `NULL`.
`scale`	Specifies the scale factor. This requires `convert = TRUE`. The default value is `NULL`.

Details

`extractPixels` extracts the pixel values from an image. The input variables are images of the same size, typically the output of a `resizeImage` transform. The output are pixel data in vector form that are typically used as features for a learner.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
train <- data.frame(Path = c(system.file("help/figures/RevolutionAnalyticslogo.png", package = "MicrosoftML"

# Loads the images from variable Path, resizes the images to 1x1 pixels and trains a neural net.
model <- rxNeuralNet(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
        resizeImage(vars = "Features", width = 1, height = 1, resizing = "Aniso"),
        extractPixels(vars = "Features")
        ),
    mlTransformVars = "Path",
    numHiddenNodes = 1,
    numIterations = 1)

# Featurizes the images from variable Path using the default model, and trains a linear model on the result.
model <- rxFastLinear(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
      resizeImage(vars = "Features", width = 224, height = 224), # If dnnModel == "AlexNet", the image has to
        extractPixels(vars = "Features"),
        featurizeImage(var = "Features")
        ),
    mlTransformVars = "Path")
```

`fastForest` *fastForest*

Description

Creates a list containing the function name and arguments to train a FastForest model with `rxEnsemble`.

Usage

```
fastForest(numTrees = 100, numLeaves = 20, minSplit = 10,
  exampleFraction = 0.7, featureFraction = 0.7, splitFraction = 0.7,
  numBins = 255, firstUsePenalty = 0, gainConfLevel = 0,
  trainThreads = 8, randomSeed = NULL, ...)
```

Arguments

`numTrees`	Specifies the total number of decision trees to create in the ensemble.By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.
`numLeaves`	The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.

`minSplit`	Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10.
`exampleFraction`	The fraction of randomly chosen instances to use for each tree. The default value is 0.7.
`featureFraction`	The fraction of randomly chosen features to use for each tree. The default value is 0.7.
`splitFraction`	The fraction of randomly chosen features to use on each split. The default value is 0.7.
`numBins`	Maximum number of distinct values (bins) per feature. The default value is 255.
`firstUsePenalty`	The feature first use penalty coefficient. The default value is 0.
`gainConfLevel`	Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.
`trainThreads`	The number of threads to use in training. If `NULL` is specified, the number of threads to use is determined internally. The default value is `NULL`.
`randomSeed`	Specifies the random seed. The default value is `NULL`.
`...`	Additional arguments.

`fastLinear` *fastLinear*

Description

Creates a list containing the function name and arguments to train a Fast Linear model with `rxEnsemble`.

Usage

```
fastLinear(lossFunction = NULL, l2Weight = NULL, l1Weight = NULL,
  trainThreads = NULL, convergenceTolerance = 0.1, maxIterations = NULL,
  shuffle = TRUE, checkFrequency = NULL, ...)
```

Arguments

`lossFunction` Specifies the empirical loss function to optimize. For binary classification, the following choices are available:

- `logLoss`: The log-loss. This is the default.
- `hingeLoss`: The SVM hinge loss. Its parameter represents the margin size.
- `smoothHingeLoss`: The smoothed hinge loss. Its parameter represents the smoothing constant.

For linear regression, squared loss `squaredLoss` is currently supported. When this parameter is set to `NULL`, its default value depends on the type of learning:

- `logLoss` for binary classification.
- `squaredLoss` for linear regression.

`l2Weight`	Specifies the L2 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.
`l1Weight`	Specifies the L1 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.
`trainThreads`	Specifies how many concurrent threads can be used to run the algorithm. When this parameter is set to `NULL`, the number of threads used is determined based on the number of logical processors available to the process as well as the sparsity of data. Set it to `1` to run the algorithm in a single thread.
`convergenceTolerance`	Specifies the tolerance threshold used as a convergence criterion. It must be between 0 and 1. The default value is `0.1`. The algorithm is considered to have converged if the relative duality gap, which is the ratio between the duality gap and the primal loss, falls below the specified convergence tolerance.
`maxIterations`	Specifies an upper bound on the number of training iterations. This parameter must be positive or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. Each iteration requires a complete pass over the training data. Training terminates after the total number of iterations reaches the specified upper bound or when the loss function converges, whichever happens earlier.
`shuffle`	Specifies whether to shuffle the training data. Set `TRUE` to shuffle the data; `FALSE` not to shuffle. The default value is `TRUE`. SDCA is a stochastic optimization algorithm. If shuffling is turned on, the training data is shuffled on each iteration.
`checkFrequency`	The number of iterations after which the loss function is computed and checked to determine whether it has converged. The value specified must be a positive integer or `NULL`. If `NULL`, the actual value is automatically computed based on data set. Otherwise, for example, if `checkFrequency = 5` is specified, then the loss function is computed and convergence is checked every 5 iterations. The computation of the loss function requires a separate complete pass over the training data.
`...`	Additional arguments.

`fastTrees` *fastTrees*

Description

Creates a list containing the function name and arguments to train a FastTree model with `rxEnsemble`.

Usage

```
fastTrees(numTrees = 100, numLeaves = 20, learningRate = 0.2,
  minSplit = 10, exampleFraction = 0.7, featureFraction = 1,
  splitFraction = 1, numBins = 255, firstUsePenalty = 0,
  gainConfLevel = 0, unbalancedSets = FALSE, trainThreads = 8,
  randomSeed = NULL, ...)
```

Arguments

`numTrees`	Specifies the total number of decision trees to create in the ensemble.By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.
`numLeaves`	The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.
`learningRate`	Determines the size of the step taken in the direction of the gradient in each step of the learning process. This determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too samll, training takes longer to converge to the best solution.
`minSplit`	Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10. Only the number of instances is counted even if instances are weighted.
`exampleFraction`	The fraction of randomly chosen instances to use for each tree. The default value is 0.7.
`featureFraction`	The fraction of randomly chosen features to use for each tree. The default value is 1.
`splitFraction`	The fraction of randomly chosen features to use on each split. The default value is 1.
`numBins`	Maximum number of distinct values (bins) per feature. If the feature has fewer values than the number indicated, each value is placed in its own bin. If there are more values, the algorithm creates `numBins` bins.
`firstUsePenalty`	The feature first use penalty coefficient. This is a form of regularization that incurs a penalty for using a new feature when creating the tree. Increase this value to create trees that don't use many features. The default value is 0.
`gainConfLevel`	Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.
`unbalancedSets`	If `TRUE`, derivatives optimized for unbalanced sets are used. Only applicable when `type` equal to `"binary"`. The default value is `FALSE`.
`trainThreads`	The number of threads to use in training. The default value is 8.
`randomSeed`	Specifies the random seed. The default value is `NULL`.
`...`	Additional arguments.

`featurizeImage` *Machine Learning Image Featurization Transform*

Description

Featurizes an image using a pre-trained deep neural network model.

Usage

```
featurizeImage(var, outVar = NULL, dnnModel = "Resnet18")
```

Arguments

`var`	Input variable containing extracted pixel values.
`outVar`	The prefix of the output variables containing the image features. If null, the input variable name will be used. The default value is `NULL`.
`dnnModel`	The pre-trained deep neural network. The possible options are: • `"resnet18"` • `"resnet50"` • `"resnet101"` • `"alexnet"` The default value is `"resnet18"`. See Deep Residual Learning for Image Recognition for details about ResNet.

Details

`featurizeImage` featurizes an image using the specified pre-trained deep neural network model. The input variables to this transforms must be extracted pixel values.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
train <- data.frame(Path = c(system.file("help/figures/RevolutionAnalyticslogo.png", package = "MicrosoftML"

# Loads the images from variable Path, resizes the images to 1x1 pixels and trains a neural net.
model <- rxNeuralNet(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
        resizeImage(vars = "Features", width = 1, height = 1, resizing = "Aniso"),
        extractPixels(vars = "Features")
        ),
    mlTransformVars = "Path",
    numHiddenNodes = 1,
    numIterations = 1)

# Featurizes the images from variable Path using the default model, and trains a linear model on the result.
model <- rxFastLinear(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
      resizeImage(vars = "Features", width = 224, height = 224), # If dnnModel == "AlexNet", the image has to
```

```
        extractPixels(vars = "Features"),
        featurizeImage(var = "Features")
        ),
    mlTransformVars = "Path")
```

getNetDefinition *Get the Net# definition from a trained neural network model*

Description

Returns the Net# definition from a trained neural network model.

Usage

```
getNetDefinition(model, getWeights = TRUE)
```

Arguments

`model`	The previously trained neural network model.
`getWeights`	If `TRUE`, the weights are included in the returned Net# definition.

Details

Returns the Net# definition from a trained neural network model. It is useful for implementing a form of continued training, where the initial weights of the model are obtained from a previously trained model. Because only the weights are initialized from the trained model (but not gradients, momentum etc.), the training is not really resumed where it was left at the end of training of the first model.

Value

A character string containing the Net# definition.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
# Train a neural network on the iris dataset for 10 iterations.
model1 <- rxNeuralNet(
    formula = Species~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    data = iris,
    numHiddenNodes=10,
    type="multi",
    numIterations=10,
    optimizer=adaDeltaSgd())

# Train another neural network on the iris dataset, initializing the topology and weights
# from the previously trained model.
model2 <- rxNeuralNet(
    formula = Species~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    data = iris,
    netDefinition=getNetDefinition(model1),
    type="multi",
    numIterations=10,
    optimizer = adaDeltaSgd())
```

`getSampleDataDir` *Get Package Sample Data Location*

Description

Location where downloaded sample data is stored.

Usage

```
getSampleDataDir(sampleDataDir = NULL, createDir = TRUE)
```

Arguments

`sampleDataDir`	Specifies the path to the location where downloaded sample data is (or is to be) stored or `NULL`.
`createDir`	`TRUE` to create the directory if it does not exist; `FALSE` not to create the directory.

Details

If `sampleDataDir` is `NULL`, the function first checks to see if an option has been set containing `sampleDataDir`, i.e. `getOption("sampleDataDir")`. If that is `NULL` too, a 'sampleDataDir' subdirectory of the current working directory is used. If `createDir` is `TRUE`, the directory is created if it does not exist.

Value

A character string containing the path to the location of the sample data.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
# This example sets the option to be the same as the default
options(sampleDataDir = file.path(getwd(), "sampleDataDir"))

dataDir <- getSampleDataDir(createDir = FALSE)
```

`getSentiment` *Machine Learning Sentiment Analyzer Transform*

Description

Scores natual language text and creates a column that contains probabilities that the sentiments in the text are positive.

Usage

```
getSentiment(vars, ...)
```

Arguments

`vars`	A character vector or list of variable names to transform. If named, the names represent the names of new variables to be created.
`...`	Additional arguments sent to compute engine.

Details

The `getSentiment` transform returns the probability that the sentiment of a natural text is positive. Currently supports only the English language.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastTrees`, `rxFastForest`, `rxNeuralNet`, `rxOneClassSvm`, `rxLogisticRegression`, `rxFastLinear`.

Examples

```
# Create the data
CustomerReviews <- data.frame(Review = c(
  "I really did not like the taste of it",
  "It was surprisingly quite good!",
  "I will never ever ever go to that place again!!"),
  stringsAsFactors = FALSE)

# Get the sentiment scores
sentimentScores <- rxFeaturize(data = CustomerReviews,
                         mlTransforms = getSentiment(vars = list(SentimentScore = "Review")))

# Let's translate the score to something more meaningful
sentimentScores$PredictedRating <- ifelse(sentimentScores$SentimentScore > 0.6,
                                          "AWESOMENESS", "BLAH")

# Let's look at the results
sentimentScores
```

`kernel` *Kernel*

Description

Kernels supported for use in computing inner products.

Usage

```
linearKernel(...)

polynomialKernel(a = NULL, bias = 0, deg = 3, ...)

rbfKernel(gamma = NULL, ...)

sigmoidKernel(gamma = NULL, coef0 = 0, ...)
```

Arguments

`a`	The numeric value for a in the term (a*<x,y> + b)^d. If not specified, `(1/(number of features)` is used.
`bias`	The numeric value for b in the term `(a*<x,y> + b)^d`.
`deg`	The integer value for d in the term `(a*<x,y> + b)^d`.
`gamma`	The numeric value for gamma in the expression `tanh(gamma*<x,y> + c)`. If not specified, `1/(number of features)` is used.
`coef0`	The numeric value for c in the expression `tanh(gamma*<x,y> + c)`.
`...`	Additional arguments passed to the Microsoft ML compute engine.

Details

These helper functions specify the kernel that is used for training in relevant algorithms. The kernals that are suppored:

- `linearKernel`: linear kernel.
- `rbfKernel`: radial basis function kernel.
- `polynomialKernel`: polynomial kernel.
- `sigmoidKernel`: sigmoid kernel.

Value

A character string defining the kernel.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Estimating the Support of a High-Dimensional Distribution

New Support Vector Algorithms

See Also

rxOneClassSvm

Examples

```
# Simulate some simple data
set.seed(7)
numRows <- 200
normalData <- data.frame(day = 1:numRows)
normalData$pageViews = runif(numRows, min = 10, max = 1000) + .5 * normalData$day
testData <- data.frame(day = 1:numRows)
# The test data has outliers above 1000
testData$pageViews = runif(numRows, min = 10, max = 1400) + .5 * testData$day

train <- function(kernelFunction, args=NULL) {
    model <- rxOneClassSvm(formula = ~pageViews + day, data = normalData,
    kernel = kernelFunction(args))
    scores <- rxPredict(model, data = testData, writeModelVars = TRUE)
    scores$groups = scores$Score > 0
    scores
}
display <- function(scores) {
    print(sum(scores$groups))
    rxLinePlot(pageViews ~ day, data = scores, groups = groups, type = "p",
     symbolColors = c("red", "blue"))
}
scores <- list()
scores$rbfKernel <- train(rbfKernel)
scores$linearKernel <- train(linearKernel)
scores$polynomialKernel <- train(polynomialKernel, (a = .2))
scores$sigmoidKernel <- train(sigmoidKernel)
display(scores$rbfKernel)
display(scores$linearKernel)
display(scores$polynomialKernel)
display(scores$sigmoidKernel)
```

---

loadImage *Machine Learning Load Image Transform*

Description

Loads image data.

Usage

```
loadImage(vars)
```

Arguments

vars	A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

Details

loadImage loads images from paths.

Value

A maml object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
train <- data.frame(Path = c(system.file("help/figures/RevolutionAnalyticslogo.png", package = "MicrosoftML"

# Loads the images from variable Path, resizes the images to 1x1 pixels and trains a neural net.
model <- rxNeuralNet(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
        resizeImage(vars = "Features", width = 1, height = 1, resizing = "Aniso"),
        extractPixels(vars = "Features")
        ),
    mlTransformVars = "Path",
    numHiddenNodes = 1,
    numIterations = 1)

# Featurizes the images from variable Path using the default model, and trains a linear model on the result.
model <- rxFastLinear(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
      resizeImage(vars = "Features", width = 224, height = 224), # If dnnModel == "AlexNet", the image has to
        extractPixels(vars = "Features"),
        featurizeImage(var = "Features")
        ),
    mlTransformVars = "Path")
```

`logisticRegression` *logisticRegression*

Description

Creates a list containing the function name and arguments to train a logistic regression model with rxEnsemble.

Usage

```
logisticRegression(l2Weight = 1, l1Weight = 1, optTol = 1e-07,
  memorySize = 20, initWtsScale = 0, maxIterations = 2147483647,
  showTrainingStats = FALSE, sgdInitTol = 0, trainThreads = NULL,
  denseOptimizer = FALSE, ...)
```

Arguments

l2Weight	The L2 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1.
l1Weight	The L1 regularization weight. Its value must be greater than or equal to 0 and the default value is set to 1.
optTol	Threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. Smaller values are slower, but more accurate. The default value is 1e-07.
memorySize	Memory size for L-BFGS, specifying the number of past positions and gradients to store for the computation of the next step. This optimization parameter limits the amount of memory that is used to compute the magnitude and direction of the next step. When you specify less memory, training is faster but less accurate. Must be greater than or equal to 1 and the default value is 20.
initWtsScale	Sets the initial weights diameter that specifies the range from which values are drawn for the initial weights. These weights are initialized randomly from within this range. For example, if the diameter is specified to be d, then the weights are uniformly distributed between -d/2 and d/2. The default value is 0, which specifies that allthe weights are initialized to 0.
maxIterations	Sets the maximum number of iterations. After this number of steps, the algorithm stops even if it has not satisfied convergence criteria.
showTrainingStats	Specify TRUE to show the statistics of training data and the trained model; otherwise, FALSE. The default value is FALSE. For additional information about model statistics, see summary.mlModel.
sgdInitTol	Set to a number greater than 0 to use Stochastic Gradient Descent (SGD) to find the initial parameters. A non-zero value set specifies the tolerance SGD uses to determine convergence. The default value is 0 specifying that SGD is not used.
trainThreads	The number of threads to use in training the model. This should be set to the number of cores on the machine. Note that L-BFGS multi-threading attempts to load dataset into memory. In case of out-of-memory issues, set trainThreads to 1 to turn off multi-threading. If NULL the number of threads to use is determined internally. The default value is NULL.
denseOptimizer	If TRUE, forces densification of the internal optimization vectors. If FALSE, enables the logistic regression optimizer use sparse or dense internal states as it finds appropriate. Setting denseOptimizer to TRUE requires the internal optimizer to use a dense internal state, which may help alleviate load on the garbage collector for some varieties of larger problems.
...	Additional arguments.

loss functions *Classification and Regression Loss functions*

Description

The loss functions for classification and regression.

Usage

```
expLoss(beta = 1, ...)

hingeLoss(margin = 1, ...)

logLoss(...)

smoothHingeLoss(smoothingConst = 1, ...)

poissonLoss(...)

squaredLoss(...)
```

Arguments

`beta`	Specifies the numeric value of beta (dilation). The default value is 1.
`margin`	Specifies the numeric margin value. The default value is 1.
`smoothingConst`	Specifies the numeric value of the smoothing constant. The default value is 1.
`...`	hidden argument.

Details

A loss function measures the discrepancy between the prediction of a machine learning algorithm and the supervised output and represents the cost of being wrong.

The classification loss functions supported are:

- `logLoss`
- `expLoss`
- `hingeLoss`
- `smoothHingeLoss`

The regression loss functions supported are:

- `poissonLoss`
- `squaredLoss`.

Value

A character string defining the loss function.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastLinear`, `rxNeuralNet`

Examples

```
train <- function(lossFunction) {

    result <- rxFastLinear(isCase ~ age + parity + education + spontaneous + induced,
                  transforms = list(isCase = case == 1), lossFunction = lossFunction,
                  data = infert,
                  type = "binary")
    coef(result)[["age"]]
}

age <- list()
age$LogLoss <- train(logLoss())
age$LogLossHinge <- train(hingeLoss())
age$LogLossSmoothHinge <- train(smoothHingeLoss())
age
```

`maOptimizer` *Optimization Algorithms*

Description

Specifies Optimization Algorithms for Neural Net.

Usage

```
adaDeltaSgd(decay = 0.95, conditioningConst = 1e-06)

sgd(learningRate = 0.001, momentum = 0, nag = FALSE, weightDecay = 0,
  lRateRedRatio = 1, lRateRedFreq = 100, lRateRedErrorRatio = 0)
```

Arguments

`decay`	Specifies the decay rate applied to gradients when calculating the step in the ADADELTA adaptive optimization algorithm. This rate is used to ensure that the learning rate continues to make progress by giving smaller weights to remote gradients in the calculation of the step size. Mathematically, it replaces the mean square of the gradients with an exponentially decaying average of the squared gradients in the denominator of the update rule. The value assigned must be in the range (0,1).
`conditioningConst`	Specifies a conditioning constant for the ADADELTA adaptive optimization algorithm that is used to condition the step size in regions where the exponentially decaying average of the squared gradients is small. The value assigned must be in the range (0,1).
`learningRate`	Specifies the size of the step taken in the direction of the negative gradient for each iteration of the learning process. The default value is `= 0.001`.
`momentum`	Specifies weights for each dimension that control the contribution of the previous step to the size of the next step during training. This modifies the `learningRate` to speed up training. The value must be `>= 0` and `< 1`.

`nag` If `TRUE`, Nesterov's Accelerated Gradient Descent is used. This method reduces the oracle complexity of gradient descent and is optimal for smooth convex optimization.

`weightDecay` Specifies the scaling weights for the step size. After each weight update, the weights in the network are scaled by `(1 -learningRate * weightDecay)`. The value must be >= `0` and < `1`.

`lRateRedRatio` Specifies the learning rate reduction ratio: the ratio by which the learning rate is reduced during training. Reducing the learning rate can avoid local minima. The value must be > `0` and <= `1`.

- A value of `1.0` means no reduction.
- A value of `0.9` means the learning rate is reduced to 90 its current value.

The reduction can be triggered either periodically, to occur after a fixed number of iterations, or when a certain error criteria concerning increases or decreases in the loss function are satisfied.

- To trigger a periodic rate reduction, specify the frequency by setting the number of iterations between reductions with the `lRateRedFreq` argument.
- To trigger rate reduction based on an error criterion, specify a number in `lRateRedErrorRatio`.

`lRateRedFreq` Sets the learning rate reduction frequency by specifying number of iterations betweeen reductions. For example, if `10` is specified, the learning rate is reduced once every 10 iterations.

`lRateRedErrorRatio`
Spefifies the learning rate reduction error criterion. If set to `0`, the learning rate is reduced if the loss increases between iterations. If set to a fractional value greater than`0`, the learning rate is reduced if the loss decreases by less than that fraction of its previous value.

Details

These functions can be used for the `optimizer` argument in `rxNeuralNet`.

- The `sgd` function specifies Stochastic Gradient Descent. maOptimizer
- The `adaDeltaSgd` function specifies the AdaDelta gradient descent, described in the 2012 paper "ADADELTA: An Adaptive Learning Rate Method" by Matthew D.Zeiler.

Value

A character string that contains the specification for the optimization algorithm.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

ADADELTA: An Adaptive Learning Rate Method

See Also

`rxNeuralNet`,

Examples

```
myIris = iris
myIris$Setosa <- iris$Species == "setosa"

res1 <- rxNeuralNet(formula = Setosa~Sepal.Length + Sepal.Width + Petal.Width,
        data = myIris,
        optimizer = sgd(learningRate = .002))

res2 <- rxNeuralNet(formula = Setosa~Sepal.Length + Sepal.Width + Petal.Width,
        data = myIris,
        optimizer = adaDeltaSgd(decay = .9, conditioningConst = 1e-05))
```

`minCount` *Feature Selection Count Mode*

Description

Count mode of feature selection used in the feature selection transform `selectFeatures`.

Usage

```
minCount(count = 1, ...)
```

Arguments

`count`	The threshold for count based feature selection. A feature is selected if and only if at least `count` examples have non-default value in the feature. The default value is 1.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

When using the count mode in feature selection transform, a feature is selected if the number of examples have at least the specified count examples of non-default values in the feature. The count mode feature selection transform is very useful when applied together with a categorical hash transform (see also, `categoricalHash`. The count feature selection can remove those features generated by hash transform that have no data in the examples.

Value

A character string defining the count mode.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`mutualInformation` `selectFeatures`

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )

    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
   mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection transform
# which selects only those hash features that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures("reviewCatHash", mode = minCount())))
```

```
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature selection transform
# which selects those features appearing with at least a count of 5.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures("reviewCatHash", mode = minCount(count = 5))))
summary(outModel3)
```

`mlDataStep` *Data step using MicrosoftML transformations*

Description

NOT YET IMPLEMENTED Data step for a data frame or RevoScaleR data source using MicrosoftML transformations

Usage

```
mlDataStep(data, outData = NULL, outVars = NULL, overwrite = FALSE,
  verbose = 1, ...)
```

Arguments

`data`	A **RevoScaleR** data source object, a data frame, or the path to a `.xdf` file.
`outData`	The output text or xdf file name, an `RxDataSource` with write capabilities, to store predictions. If `NULL`, a data frame will be returned.
`outVars`	specifies the variables to retain in the output data set.
`overwrite`	logical value. If `TRUE`, an existing `outData` will be overwritten.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Value

A data frame or an `RxDataSource` object representing the created output data.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxPredict.mlModel`.

Examples

```
# Not yet implemented
```

`mutualInformation` *Feature Selection Mutual Information Mode*

Description

Mutual information mode of feature selection used in the feature selection transform `selectFeatures`.

Usage

```
mutualInformation(numFeaturesToKeep = 1000, numBins = 256, ...)
```

Arguments

`numFeaturesToKeep`	If the number of features to keep is specified to be `n`, the transform picks the `n` features that have the highest mutual information with the dependent variable. The default value is 1000.
`numBins`	Maximum number of bins for numerical values. Powers of 2 are recommended. The default value is 256.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The mutual information of two random variables `X` and `Y` is a measure of the mutual dependence between the variables. Formally, the mutual information can be written as:

`I(X;Y) = E[log(p(x,y)) - log(p(x)) - log(p(y))]`

where the expectation is taken over the joint distribution of `X` and `Y`. Here `p(x,y)` is the joint probability density function of `X` and `Y`, `p(x)` and `p(y)` are the marginal probability density functions of `X` and `Y` respectively. In general, a higher mutual information between the dependent variable (or label) and an independent varialbe (or feature) means that the label has higher mutual dependence over that feature.

The mutual information feature selection mode selects the features based on the mutual information. It keeps the top `numFeaturesToKeep` features with the largest mutual information with the label.

Value

a character string defining the mode.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: Mutual Information

See Also

`minCount` `selectFeatures`

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )

    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
   mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection transform
# which selects only those hash features that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures("reviewCatHash", mode = minCount())))
```

```
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature selection transform
# which selects those features appearing with at least a count of 5.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures("reviewCatHash", mode = minCount(count = 5))))
summary(outModel3)
```

`neuralNet` *neuralNet*

Description

Creates a list containing the function name and arguments to train a NeuralNet model with `rxEnsemble`.

Usage

```
neuralNet(numHiddenNodes = 100, numIterations = 100, optimizer = sgd(),
  netDefinition = NULL, initWtsDiameter = 0.1, maxNorm = 0,
  acceleration = c("sse", "gpu"), miniBatchSize = 1, ...)
```

Arguments

`numHiddenNodes`	The default number of hidden nodes in the neural net. The default value is 100.
`numIterations`	The number of iterations on the full training set. The default value is 100.
`optimizer`	A list specifying either the `sgd` or `adaptive` optimization algorithm. This list can be created using `sgd` or `adaDeltaSgd`. The default value is sgd.
`netDefinition`	The Net# definition of the structure of the neural network. For more information about the Net# language, see Reference Guide
`initWtsDiameter`	Sets the initial weights diameter that specifies the range from which values are drawn for the initial learning weights. The weights are initialized randomly from within this range. The default value is 0.1.
`maxNorm`	Specifies an upper bound to constrain the norm of the incoming weight vector at each hidden unit. This can be very important in maxout neural networks as well as in cases where training produces unbounded weights.
`acceleration`	Specifies the type of hardware acceleration to use. Possible values are "sse" and "gpu". For GPU acceleration, it is recommended to use a miniBatchSize greater than one. If you want to use the GPU acceleration, there are additional manual setup steps are required: • Download and install NVidia CUDA Toolkit 6.5 (CUDA Toolkit). • Download and install NVidia cuDNN v2 Library (cudnn Library). • Find the libs directory of the MicrosoftRML package by calling `system.file("mxLibs/x64",` • Copy cublas64_65.dll, cudart64_65.dll and cusparse64_65.dll from the CUDA Toolkit 6.5 into the libs directory of the MicrosoftML package. • Copy cudnn64_65.dll from the cuDNN v2 Library into the libs directory of the MicrosoftML package.

`miniBatchSize`	Sets the mini-batch size. Recommended values are between 1 and 256. This parameter is only used when the acceleration is GPU. Setting this parameter to a higher value improves the speed of training, but it might negatively affect the accuracy. The default value is 1.
`...`	Additional arguments.

`ngram` *Machine Learning Feature Extractors*

Description

Feature Extractors that can be used with mtText.

Usage

```
ngramCount(ngramLength = 1, skipLength = 0, maxNumTerms = 1e+07,
  weighting = "tf")

ngramHash(ngramLength = 1, skipLength = 0, hashBits = 16,
  seed = 314489979, ordered = TRUE, invertHash = 0)
```

Arguments

`ngramLength`	An integer that specifies the maximum number of tokens to take when constructing an n-gram. The default value is 1.
`skipLength`	An integer that specifies the maximum number of tokens to skip when constructing an n-gram. If the value specified as skip length is `k`, then n-grams can contain up to k skips (not necessarily consecutive). For example, if `k=2`, then the 3-grams extracted from the text "the sky is blue today" are: "the sky is", "the sky blue", "the sky today", "the is blue", "the is today" and "the blue today". The default value is 0.
`maxNumTerms`	An integer that specifies the maximum number of categories to include in the dictionary. The default value is 10000000.
`weighting`	A character string that specifies the weighting criteria: • `"tf"`: to use term frequency. • `"idf"`: to use inverse document frequency. • `"tfidf"`: to use both term frequency and inverse document frequency.
`hashBits`	integer value. Number of bits to hash into. Must be between 1 and 30, inclusive.
`seed`	integer value. Hashing seed.
`ordered`	`TRUE` to include the position of each term in the hash. Otherwise, `FALSE`. The default value is `TRUE`.
`invertHash`	An integer specifying the limit on the number of keys that can be used to generate the slot name. `0` means no invert hashing; `-1` means no limit. While a zero value gives better performance, a non-zero value is needed to get meaningful coefficent names.

Details

ngramCount allows defining arguments for count-based feature extraction. It accepts following options: ngramLenght, skipLenght, maxNumTerms and weighting.

ngramHash allows defining arguments for hashing-based feature extraction. It accepts the following options: ngramLenght, skipLenght, hashBits, seed, ordered and invertHash.

Value

A character string defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

featurizeText.

Examples

```
 myData <- data.frame(opinion = c(
    "I love it!",
    "I love it!",
    "Love it!",
    "I love it a lot!",
    "Really love it!",
    "I hate it",
    "I hate it",
    "I hate it.",
    "Hate it",
    "Hate"),
    like = rep(c(TRUE, FALSE), each = 5),
    stringsAsFactors = FALSE)

outModel1 <- rxLogisticRegression(like~opinionCount, data = myData,
    mlTransforms = list(featurizeText(vars = c(opinionCount = "opinion"),
        wordFeatureExtractor = ngramHash(invertHash = -1, hashBits = 3))))
summary(outModel1)

outModel2 <- rxLogisticRegression(like~opinionCount, data = myData,
    mlTransforms = list(featurizeText(vars = c(opinionCount = "opinion"),
        wordFeatureExtractor = ngramCount(maxNumTerms = 5, weighting = "tf"))))
summary(outModel2)
```

---

oneClassSvm *oneClassSvm*

Description

Creates a list containing the function name and arguments to train a OneClassSvm model with rxEnsemble.

Usage

```
oneClassSvm(cacheSize = 100, kernel = rbfKernel(), epsilon = 0.001,
  nu = 0.1, shrink = TRUE, ...)
```

Arguments

`cacheSize`	The maximal size in MB of the cache that stores the training data. Increase this for large training sets. The default value is 100 MB.
`kernel`	A character string representing the kernel used for computing inner products. For more information, see `maKernel`. The following choices are available: • `rbfKernel()`: Radial basis function kernel. It's parameter represents`gamma` in the term `exp(-gamma|x-y|^2`. If not specified, it defaults to `1` divided by the number of features used. For example, `rbfKernel(gamma = .1)`. This is the default value. • `linearKernel()`: Linear kernel. • `polynomialKernel()`: Polynomial kernel with parameter names `a`, `bias`, and `deg` in the term `(a*<x,y> + bias)^deg`. The `bias`, defaults to `0`. The degree, `deg`, defaults to `3`. If `a` is not specified, it is set to `1` divided by the number of features. For example, `maKernelPoynomial(bias = 0, deg = 3)`. • `sigmoidKernel()`: Sigmoid kernel with parameter names `gamma` and `coef0` in the term `tanh(gamma*<x,y> + coef0)`. `gamma`, defaults to to `1` divided by the number of features. The parameter `coef0` defaults to `0`. For example, `sigmoidKernel(gamma = .1, coef0 = 0)`.
`epsilon`	The threshold for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. The value must be greater than or equal to `.Machine$double.eps`. The default value is 0.001.
`nu`	The trade-off between the fraction of outliers and the number of support vectors (represented by the Greek letter nu). Must be between 0 and 1, typically between 0.1 and 0.5. The default value is 0.1.
`shrink`	Uses the shrinking heuristic if `TRUE`. In this case, some samples will be "shrunk" during the training procedure, which may speed up training. The default value is `TRUE`.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

`resizeImage` *Machine Learning Resize Image Transform*

Description

Resizes an image to a specified dimension using a specified resizing method.

Usage

```
resizeImage(vars, width = 224, height = 224, resizingOption = "IsoCrop")
```

Arguments

`vars`	A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.
`width`	Specifies the width of the scaled image in pixels. The default value is 224.
`height`	Specifies the height of the scaled image in pixels. The default value is 224.
`resizingOption`	Specified the resizing method to use. Note that all methods are using bilinear interpolation. The options are: • `"IsoPad"`: The image is resized such that the aspect ratio is preserved. If needed, the image is padded with black to fit the new width or height. • `"IsoCrop"`: The image is resized such that the aspect ratio is preserved. If needed, the image is cropped to fit the new width or height. • `"Aniso"`: The image is stretched to the new width and height, without preserving the aspect ratio. The default value is `"IsoPad"`.

Details

`resizeImage` resizes an image to the specified height and width using a specified resizing method. The input variables to this transforms must be images, typically the result of the `loadImage` transform.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

Examples

```
train <- data.frame(Path = c(system.file("help/figures/RevolutionAnalyticslogo.png", package = "MicrosoftML"

# Loads the images from variable Path, resizes the images to 1x1 pixels and trains a neural net.
model <- rxNeuralNet(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
        resizeImage(vars = "Features", width = 1, height = 1, resizing = "Aniso"),
        extractPixels(vars = "Features")
        ),
    mlTransformVars = "Path",
    numHiddenNodes = 1,
    numIterations = 1)

# Featurizes the images from variable Path using the default model, and trains a linear model on the result.
model <- rxFastLinear(
    Label ~ Features,
    data = train,
    mlTransforms = list(
        loadImage(vars = list(Features = "Path")),
      resizeImage(vars = "Features", width = 224, height = 224), # If dnnModel == "AlexNet", the image has to
        extractPixels(vars = "Features"),
        featurizeImage(var = "Features")
        ),
    mlTransformVars = "Path")
```

`rxEnsemble` *Ensembles*

Description

Train an ensemble of models

Usage

```
rxEnsemble(formula = NULL, data, trainers, type = c("binary", "regression",
  "multiClass", "anomaly"), randomSeed = NULL,
  modelCount = length(trainers), replace = FALSE, sampRate = NULL,
  splitData = FALSE, combineMethod = c("median", "average", "vote"),
  maxCalibration = 1e+05, mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"), ...)
```

Arguments

`formula`	The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.
`data`	A data source object or a character string specifying a '`.xdf`' file or a data frame object. Alternatively, it can be a list of data sources indicating each model should be trained using one of the data sources in the list. In this case, the length of the data list must be equal to `modelCount`.
`trainers`	A list of trainers with their arguments. The trainers are created by using `fastTrees`, `fastForest`, `fastLinear`, `logisticRegression` or `neuralNet`.
`type`	A character string that specifies the type of ensemble: `"binary"` for Binary Classification or `"regression"` for Regression.
`randomSeed`	Specifies the random seed. The default value is `NULL`.
`modelCount`	Specifies the number of models to train. If this number is greater than the length of the trainers list, the trainers list is duplicated to match `modelCount`.
`replace`	A logical value specifying if the sampling of observations should be done with or without replacement. The default value is /codeFALSE.
`sampRate`	a scalar of positive value specifying the percentage of observations to sample for each trainer. The default is 1.0 for sampling with replacement (i.e., replace=TRUE) and 0.632 for sampling without replacement (i.e., replace=FALSE). When splitData is TRUE, the default of sampRate is 1.0 (no sampling is done before splitting).

splitData	A logical value specifying whether or not to train the base models on non-overlapping partitions. The default is FALSE. It is available only for RxSpark compute context and ignored for others.
combineMethod	Specifies the method used to combine the models: • median to compute the median of the individual model outputs, • average to compute the average of the individual model outputs and • vote to compute (pos-neg) / the total number of models, where 'pos' is the number of positive outputs and 'neg' is the number of negative outputs.
maxCalibration	Specifies the maximum number of examples to use for calibration. This argument is ignored for all tasks other than binary classification.
mlTransforms	Specifies a list of MicrosoftML transforms to be performed on the data before training or NULL if no transforms are to be performed. Transforms that require an additional pass over the data (such as featurizeText, categorical) are not allowed. These transformations are performed after any specified R transformations. The default value is NULL.
mlTransformVars	Specifies a character vector of variable names to be used in mlTransforms or NULL if none are to be used. The default value is NULL.
rowSelection	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, rowSelection = "old" will only use observations in which the value of the variable old is TRUE. rowSelection = (age > 20 only uses observations in which the value of the age variable is between 20 and 65 and the value of the log of the income variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments transforms or transformFunc). As with all expressions, rowSelection can be defined outside of the function call using the expression function.
transforms	An expression of the form list(name = expression, ...) that represents the first round of variable transformations. As with all expressions, transforms (or rowSelection) can be defined outside of the function call using the expression function. The default value is NULL.
transformObjects	A named list that contains objects that can be referenced by transforms, transformsFunc, and rowSelection. The default value is NULL.
transformFunc	The variable transformation function. See rxTransform for details. The default value is NULL.
transformVars	A character vector of input data set variables needed for the transformation function. See rxTransform for details. The default value is NULL.
transformPackages	A character vector specifying additional R packages (outside of those specified in rxGetOption("transformPackages")) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their transforms and transformFunc arguments or those defined implicitly via their formula or rowSelection arguments. The transformPackages argument may also be NULL, indicating that no packages outside rxGetOption("transformPackages") are preloaded. The default value is NULL.
transformEnvir	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If transformEnvir = NULL, a new "hash" environment with parent baseenv() is used instead. The default value is NULL.

`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information. The default value is `1`.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxSpark` compute contexts are supported. When `RxSpark` is specified, the training of the models is done in a distributed way, and the ensembling is done locally. Note that the compute context cannot be non-waiting.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

/coderxEnsemble is a function that trains a number of models of various kinds to obtain better predictive performance than could be obtained from a single model.

Value

A `rxEnsemble` object with the trained ensemble model.

Examples

```
# Create an ensemble of regression rxFastTrees models

# use xdf data source
dataFile <- file.path(rxGetOption("sampleDataDir"), "claims4blocks.xdf")
rxGetInfo(dataFile, getVarInfo = TRUE, getBlockSizes = TRUE)
form <- cost ~ age + type + number

rxSetComputeContext("localpar")
rxGetComputeContext()

# build an ensemble model that contains three 'rxFastTrees' models with different parameters
ensemble <- rxEnsemble(
    formula = form,
    data = dataFile,
    type = "regression",
   trainers = list(fastTrees(), fastTrees(numTrees = 60), fastTrees(learningRate = 0.1)), #a list of trainers
    replace = TRUE # Indicates using a bootstrap sample for each trainer
    )

# use text data source
colInfo <- list(DayOfWeek = list(type = "factor", levels = c("Monday", "Tuesday", "Wednesday", "Thursday", "F

source <- system.file("SampleData/AirlineDemoSmall.csv", package = "RevoScaleR")
data <- RxTextData(source, missingValueString = "M", colInfo = colInfo)
```

```
# When 'distributed' is TRUE distributed data source is created
distributed <- FALSE
if (distributed) {
    bigDataDirRoot <- "/share"
    inputDir <- file.path(bigDataDirRoot, "AirlineDemoSmall")
    rxHadoopMakeDir(inputDir)
    rxHadoopCopyFromLocal(source, inputDir)
    hdfsFS <- RxHdfsFileSystem()
   data <- RxTextData(file = inputDir, missingValueString = "M", colInfo = colInfo, fileSystem = hdfsFS)
}

# When 'distributed' is TRUE training is distributed
if (distributed) {
    cc <- rxSetComputeContext(RxSpark())
} else {
    cc <- rxGetComputeContext()
}

ensemble <- rxEnsemble(
    formula = ArrDelay ~ DayOfWeek,
    data = data,
    type = "regression",
   trainers = list(fastTrees(), fastTrees(numTrees = 60), fastTrees(learningRate = 0.1)), # The ensemble will
    replace = TRUE # Indicates using a bootstrap sample for each trainer
    )

# Change the compute context back to previous for scoring
rxSetComputeContext(cc)

# Put score and model variables in data frame
scores <- rxPredict(ensemble, data = data, writeModelVars = TRUE)

# Plot actual versus predicted values with smoothed line
rxLinePlot(Score ~ ArrDelay, type = c("p", "smooth"), data = scores)
```

rxFastForest *Fast Forest*

Description

Machine Learning Fast Forest

Usage

```
rxFastForest(formula = NULL, data, type = c("binary", "regression"),
  numTrees = 100, numLeaves = 20, minSplit = 10, exampleFraction = 0.7,
  featureFraction = 0.7, splitFraction = 0.7, numBins = 255,
  firstUsePenalty = 0, gainConfLevel = 0, trainThreads = 8,
  randomSeed = NULL, mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 2,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula`	The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.
`data`	A data source object or a character string specifying a '`.xdf`' file or a data frame object.
`type`	A character string denoting Fast Tree type: • `"binary"` for the default Fast Tree Binary Classification or • `"regression"` for Fast Tree Regression.
`numTrees`	Specifies the total number of decision trees to create in the ensemble.By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.
`numLeaves`	The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.
`minSplit`	Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10.
`exampleFraction`	The fraction of randomly chosen instances to use for each tree. The default value is 0.7.
`featureFraction`	The fraction of randomly chosen features to use for each tree. The default value is 0.7.
`splitFraction`	The fraction of randomly chosen features to use on each split. The default value is 0.7.
`numBins`	Maximum number of distinct values (bins) per feature. The default value is 255.
`firstUsePenalty`	The feature first use penalty coefficient. The default value is 0.
`gainConfLevel`	Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.
`trainThreads`	The number of threads to use in training. If `NULL` is specified, the number of threads to use is determined internally. The default value is `NULL`.
`randomSeed`	Specifies the random seed. The default value is `NULL`.
`mlTransforms`	Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.
`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.

`rowSelection` Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.

`transforms` An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.

`transformObjects`
A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc` The variable transformation function. See rxTransform for details.

`transformVars` A character vector of input data set variables needed for the transformation function. See rxTransform for details.

`transformPackages`
A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.

`transformEnvir` A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.

`blocksPerRead` Specifies the number of blocks to read for each chunk of data read from the data source.

`reportProgress` An integer value that specifies the level of reporting on the row processing progress:

- `0`: no progress is reported.
- `1`: the number of processed rows is printed and updated.
- `2`: rows processed and timings are reported.
- `3`: rows processed and all timings are reported.

`verbose` An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.

`computeContext` Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.

`ensemble` Control parameters for ensembling.

`...` Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

Decision trees are non-parametric models that perform a sequence of simple tests on inputs. This decision procedure maps them to outputs found in the training dataset whose inputs were similar to the instance being processed. A decision is made at each node of the binary tree data structure based on a measure of similarity that maps each instance recursively through the branches of the tree until the appropriate leaf node is reached and the output decision returned.

Decision trees have several advantages:

- They are efficient in both computation and memory usage during training and prediction.
- They can represent non-linear decision boundaries.
- They perform integrated feature selection and classification.
- They are resilient in the presence of noisy features.

Fast forest regression is a random forest and quantile regression forest implementation using the regression tree learner in `rxFastTrees`. The model consists of an ensemble of decision trees. Each tree in a decision forest outputs a Gaussian distribution by way of prediction. An aggregation is performed over the ensemble of trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

This decision forest classifier consists of an ensemble of decision trees. Generally, ensemble models provide better coverage and accuracy than single decision trees. Each tree in a decision forest outputs a Gaussian distribution by way of prediction. An aggregation is performed over the ensemble of trees to find a Gaussian distribution closest to the combined distribution for all trees in the model.

Value

- `rxFastForest`: A `rxFastForest` object with the trained model.
- `FastForest`: A learner specification object of class `maml` for the Fast Forest trainer.

Note

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: Random forest

Quantile regression forest

From Stumps to Trees to Forests

See Also

`rxFastTrees`, `rxFastLinear`, `rxLogisticRegression`, `rxNeuralNet`, `rxOneClassSvm`, `featurizeText`, `categorical`, `categoricalHash`, `rxPredict.mlModel`.

Examples

```
# Estimate a binary classification forest
infert1 <- infert
infert1$isCase = (infert1$case == 1)
forestModel <- rxFastForest(formula = isCase ~ age + parity + education + spontaneous + induced,
        data = infert1)

# Create text file with per-instance results using rxPredict
txtOutFile <- tempfile(pattern = "scoreOut", fileext = ".txt")
txtOutDS <- RxTextData(file = txtOutFile)
scoreDS <- rxPredict(forestModel, data = infert1,
   extraVarsToWrite = c("isCase", "Score"), outData = txtOutDS)

# Print the fist ten rows
rxDataStep(scoreDS, numRows = 10)

# Clean-up
file.remove(txtOutFile)

######################################################################
# Estimate a regression fast forest

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Forest for train data
rxFastForestReg <- rxFastForest(airFormula, type = "regression",
    data = trainAir)

# Put score and model variables in data frame
rxFastForestScoreDF <- rxPredict(rxFastForestReg, data = testAir,
    writeModelVars = TRUE)

# Plot actual versus predicted values with smoothed line
rxLinePlot(Score ~ Ozone, type = c("p", "smooth"), data = rxFastForestScoreDF)
```

`rxFastLinear` *Fast Linear Model – Stochastic Dual Coordinate Ascent*

Description

A Stochastic Dual Coordinate Ascent (SDCA) optimization trainer for linear binary classification and regression.

`rxFastLinear` is a trainer based on the Stochastic Dual Coordinate Ascent (SDCA) method, a state-of-the-art optimization technique for convex objective functions. The algorithm can be scaled for use on large out-of-memory data sets due to a semi-asynchronized implementation that supports multi-threading. primal and dual updates in a separate thread. Several choices of loss functions are also provided. The SDCA method combines several of the best properties and capabilities of logistic regression and SVM algorithms. For more information on SDCA, see the citations in the reference section.

Traditional optimization algorithms, such as stochastic gradient descent (SGD), optimize the empirical loss function directly. The SDCA chooses a different approach that optimizes the dual problem instead. The dual loss function is parametrized by per-example weights. In each iteration, when a training example from the training data set is read, the corresponding example weight is adjusted so that the dual loss function is optimized with respect to the current example. No learning rate is needed by SDCA to determine step size as is required by various gradient descent methods.

`rxFastLinear` supports binary classification with three types of loss functions currently: Log loss, hinge loss, and smoothed hinge loss. Linear regression also supports with squared loss function. Elastic net regularization can be specified by the `l2Weight` and `l1Weight` parameters. Note that the `l2Weight` has an effect on the rate of convergence. In general, the larger the `l2Weight`, the faster SDCA converges.

Note that `rxFastLinear` is a stochastic and streaming optimization algorithm. The results depends on the order of the training data. For reproducible results, it is recommended that one sets `shuffle` to `FALSE` and `trainThreads` to `1`.

Usage

```
rxFastLinear(formula = NULL, data, type = c("binary", "regression"),
  lossFunction = NULL, l2Weight = NULL, l1Weight = NULL,
  trainThreads = NULL, convergenceTolerance = 0.1, maxIterations = NULL,
  shuffle = TRUE, checkFrequency = NULL, normalize = "auto",
  mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
  transforms = NULL, transformObjects = NULL, transformFunc = NULL,
  transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula` The formula described in `rxFormula`. Interaction terms and `F()` are currently not supported in **MicrosoftML**.

`data` A data source object or a character string specifying a '`.xdf`' file or a data frame object.

`type` Specifies the model type with a character string: `"binary"` for the default binary classification or `"regression"` for linear regression.

`lossFunction` Specifies the empirical loss function to optimize. For binary classification, the following choices are available:

- `logLoss`: The log-loss. This is the default.
- `hingeLoss`: The SVM hinge loss. Its parameter represents the margin size.
- `smoothHingeLoss`: The smoothed hinge loss. Its parameter represents the smoothing constant.

For linear regression, squared loss `squaredLoss` is currently supported. When this parameter is set to `NULL`, its default value depends on the type of learning:

- `logLoss` for binary classification.
- `squaredLoss` for linear regression.

`l2Weight`	Specifies the L2 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.
`l1Weight`	Specifies the L1 regularization weight. The value must be either non-negative or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. `NULL` is the default value.
`trainThreads`	Specifies how many concurrent threads can be used to run the algorithm. When this parameter is set to `NULL`, the number of threads used is determined based on the number of logical processors available to the process as well as the sparsity of data. Set it to `1` to run the algorithm in a single thread.
`convergenceTolerance`	Specifies the tolerance threshold used as a convergence criterion. It must be between 0 and 1. The default value is `0.1`. The algorithm is considered to have converged if the relative duality gap, which is the ratio between the duality gap and the primal loss, falls below the specified convergence tolerance.
`maxIterations`	Specifies an upper bound on the number of training iterations. This parameter must be positive or `NULL`. If `NULL` is specified, the actual value is automatically computed based on data set. Each iteration requires a complete pass over the training data. Training terminates after the total number of iterations reaches the specified upper bound or when the loss function converges, whichever happens earlier.
`shuffle`	Specifies whether to shuffle the training data. Set `TRUE` to shuffle the data; `FALSE` not to shuffle. The default value is `TRUE`. SDCA is a stochastic optimization algorithm. If shuffling is turned on, the training data is shuffled on each iteration.
`checkFrequency`	The number of iterations after which the loss function is computed and checked to determine whether it has converged. The value specified must be a positive integer or `NULL`. If `NULL`, the actual value is automatically computed based on data set. Otherwise, for example, if `checkFrequency = 5` is specified, then the loss function is computed and convergence is checked every 5 iterations. The computation of the loss function requires a separate complete pass over the training data.
`normalize`	Specifies the type of automatic normalization used: • `"auto"`: if normalization is needed, it is automatically performed. This is the default value. • `"no"`: no normalization is performed. • `"yes"`: normalization is performed. • `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed. Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval [a, b] where `-1 <= a <= 0` and `0 <= b <= 1` and `b - a = 1`. This normalizer preserves sparsity by mapping zero to zero.
`mlTransforms`	Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.
`rowSelection`	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.
`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.
`transformFunc`	The variable transformation function. See rxTransform for details.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`ensemble`	Control parameters for ensembling.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Value

- `rxFastLinear`: A `rxFastLinear` object with the trained model.
- `FastLinear`: A learner specification object of class `maml` for the Fast Linear trainer.

Note

This algorithm is multi-threaded and will not attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Scaling Up Stochastic Dual Coordinate Ascent

Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization

See Also

`logLoss`, `hingeLoss`, `smoothHingeLoss`, `squaredLoss`, `rxFastTrees`, `rxFastForest`, `rxLogisticRegression`, `rxNeuralNet`, `rxOneClassSvm`, `featurizeText`, `categorical`, `categoricalHash`, `rxPredict.mlModel`.

Examples

```
# Train a binary classiication model with rxFastLinear
res1 <- rxFastLinear(isCase ~ age + parity + education + spontaneous + induced,
                  transforms = list(isCase = case == 1),
                  data = infert,
                  type = "binary")
# Print a summary of the model
summary(res1)

# Score to a data frame
scoreDF <- rxPredict(res1, data = infert,
    extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data = scoreDF)
plot(roc1)
rxAuc(roc1)

#########################################################################
# rxFastLinear Regression

# Create an xdf file with the attitude data
myXdf <- tempfile(pattern = "tempAttitude", fileext = ".xdf")
rxDataStep(attitude, myXdf, rowsPerRead = 50, overwrite = TRUE)
myXdfDS <- RxXdfData(file = myXdf)

attitudeForm <- rating ~ complaints + privileges + learning +
    raises + critical + advance

# Estimate a regression model with rxFastLinear
res2 <- rxFastLinear(formula = attitudeForm,  data = myXdfDS,
    type = "regression")
```

```
# Score to data frame
scoreOut2 <- rxPredict(res2, data = myXdfDS,
    extraVarsToWrite = "rating")

# Plot the rating versus the score with a regression line
rxLinePlot(rating~Score, type = c("p","r"), data = scoreOut2)

# Clean up
file.remove(myXdf)
```

---

rxFastTrees *Fast Tree*

Description

Machine Learning Fast Tree

Usage

```
rxFastTrees(formula = NULL, data, type = c("binary", "regression"),
  numTrees = 100, numLeaves = 20, learningRate = 0.2, minSplit = 10,
  exampleFraction = 0.7, featureFraction = 1, splitFraction = 1,
  numBins = 255, firstUsePenalty = 0, gainConfLevel = 0,
  unbalancedSets = FALSE, trainThreads = 8, randomSeed = NULL,
  mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
  transforms = NULL, transformObjects = NULL, transformFunc = NULL,
  transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 2,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula`	The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.
`data`	A data source object or a character string specifying a '`.xdf`' file or a data frame object.
`type`	A character string that specifies the type of Fast Tree: `"binary"` for the default Fast Tree Binary Classification or `"regression"` for Fast Tree Regression.
`numTrees`	Specifies the total number of decision trees to create in the ensemble.By creating more decision trees, you can potentially get better coverage, but the training time increases. The default value is 100.
`numLeaves`	The maximum number of leaves (terminal nodes) that can be created in any tree. Higher values potentially increase the size of the tree and get better precision, but risk overfitting and requiring longer training times. The default value is 20.
`learningRate`	Determines the size of the step taken in the direction of the gradient in each step of the learning process. This determines how fast or slow the learner converges on the optimal solution. If the step size is too big, you might overshoot the optimal solution. If the step size is too samll, training takes longer to converge to the best solution.

`minSplit`	Minimum number of training instances required to form a leaf. That is, the minimal number of documents allowed in a leaf of a regression tree, out of the sub-sampled data. A 'split' means that features in each level of the tree (node) are randomly divided. The default value is 10. Only the number of instances is counted even if instances are weighted.
`exampleFraction`	The fraction of randomly chosen instances to use for each tree. The default value is 0.7.
`featureFraction`	The fraction of randomly chosen features to use for each tree. The default value is 1.
`splitFraction`	The fraction of randomly chosen features to use on each split. The default value is 1.
`numBins`	Maximum number of distinct values (bins) per feature. If the feature has fewer values than the number indicated, each value is placed in its own bin. If there are more values, the algorithm creates `numBins` bins.
`firstUsePenalty`	The feature first use penalty coefficient. This is a form of regularization that incurs a penalty for using a new feature when creating the tree. Increase this value to create trees that don't use many features. The default value is 0.
`gainConfLevel`	Tree fitting gain confidence requirement (should be in the range [0,1)). The default value is 0.
`unbalancedSets`	If `TRUE`, derivatives optimized for unbalanced sets are used. Only applicable when `type` equal to `"binary"`. The default value is `FALSE`.
`trainThreads`	The number of threads to use in training. The default value is 8.
`randomSeed`	Specifies the random seed. The default value is `NULL`.
`mlTransforms`	Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.
`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.
`rowSelection`	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.
`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.

`transformFunc`	The variable transformation function. See rxTransform for details.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`ensemble`	Control parameters for ensembling.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

rxFastTrees is an implementation of FastRank. FastRank is an efficient implementation of the MART gradient boosting algorithm. Gradient boosting is a machine learning technique for regression problems. It builds each regression tree in a step-wise fashion, using a predefined loss function to measure the error for each step and corrects for it in the next. So this prediction model is actually an ensemble of weaker prediction models. In regression problems, boosting builds a series of of such trees in a step-wise fashion and then selects the optimal tree using an arbitrary differentiable loss function.

MART learns an ensemble of regression trees, which is a decision tree with scalar values in its leaves. A decision (or regression) tree is a binary tree-like flow chart, where at each interior node one decides which of the two child nodes to continue to based on one of the feature values from the input. At each leaf node, a value is returned. In the interior nodes, the decision is based on the test `"x <= v"`, where `x` is the value of the feature in the input sample and `v` is one of the possible values of this feature. The functions that can be produced by a regression tree are all the piece-wise constant functions.

The ensemble of trees is produced by computing, in each step, a regression tree that approximates the gradient of the loss function, and adding it to the previous tree with coefficients that minimize the loss of the new tree. The output of the ensemble produced by MART on a given instance is the sum of the tree outputs.

- In case of a binary classification problem, the output is converted to a probability by using some form of calibration.
- In case of a regression problem, the output is the predicted value of the function.
- In case of a ranking problem, the instances are ordered by the output value of the ensemble.

If `type` is set to `"regression"`, a regression version of FastTree is used. If set to `"ranking"`, a ranking version of FastTree is used. In the ranking case, the instances should be ordered by the output of the tree ensemble. The only difference in the settings of these versions is in the calibration settings, which are needed only for classification.

Value

- `rxFastTrees`: A `rxFastTrees` object with the trained model.
- `FastTree`: A learner specification object of class `maml` for the Fast Tree trainer.

Note

This algorithm is multi-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: Gradient boosting (Gradient tree boosting)

Greedy function approximation: A gradient boosting machine.

See Also

`rxFastForest`, `rxFastLinear`, `rxLogisticRegression`, `rxNeuralNet`, `rxOneClassSvm`, `featurizeText`, `categorical`, `categoricalHash`, `rxPredict.mlModel`.

Examples

```
# Estimate a binary classification tree
infert1 <- infert
infert1$isCase = (infert1$case == 1)
treeModel <- rxFastTrees(formula = isCase ~ age + parity + education + spontaneous + induced,
        data = infert1)

# Create xdf file with per-instance results using rxPredict
xdfOut <- tempfile(pattern = "scoreOut", fileext = ".xdf")
scoreDS <- rxPredict(treeModel, data = infert1,
   extraVarsToWrite = c("isCase", "Score"),
   outData = xdfOut)

rxDataStep(scoreDS, numRows = 10)

# Clean-up
file.remove(xdfOut)
```

```
######################################################################
# Estimate a regression fast tree

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Tree for train data
fastTreeReg <- rxFastTrees(airFormula, type = "regression",
    data = trainAir)

# Put score and model variables in data frame
fastTreeScoreDF <- rxPredict(fastTreeReg, data = testAir,
    writeModelVars = TRUE)

# Plot actual versus predicted values with smoothed line
rxLinePlot(Score ~ Ozone, type = c("p", "smooth"), data = fastTreeScoreDF)
```

rxFeaturize *Data Transformation for RevoScaleR data sources*

Description

Transforms data from an input data set to an output data set.

Usage

```
rxFeaturize(data, outData = NULL, overwrite = FALSE, dataThreads = NULL,
  randomSeed = NULL, maxSlots = 5000, mlTransforms = NULL,
  mlTransformVars = NULL, rowSelection = NULL, transforms = NULL,
  transformObjects = NULL, transformFunc = NULL, transformVars = NULL,
  transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"), ...)
```

Arguments

`data`	A **RevoScaleR** data source object, a data frame, or the path to a `.xdf` file.
`outData`	Output text or xdf file name or an `RxDataSource` with write capabilities in which to store transformed data. If `NULL`, a data frame is returned. The default value is `NULL`.
`overwrite`	If `TRUE`, an existing `outData` is overwritten; if `FALSE` an existing `outData` is not overwritten. The default value is /codeFALSE.
`dataThreads`	An integer specifying the desired degree of parallelism in the data pipeline. If `NULL`, the number of threads used is determined internally. The default value is `NULL`.
`randomSeed`	Specifies the random seed. The default value is `NULL`.

`maxSlots`	Max slots to return for vector valued columns (<=0 to return all).
`mlTransforms`	Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.
`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.
`rowSelection`	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.
`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function. The default value is `NULL`.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`. The default value is `NULL`.
`transformFunc`	The variable transformation function. See rxTransform for details. The default value is `NULL`.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details. The default value is `NULL`.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded. The default value is `NULL`.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead The default value is `NULL`.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported.

	• 3: rows processed and all timings are reported. The default value is 1.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from 1 to 4 provide increasing amounts of information. The default value is 1.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Value

A data frame or an `RxDataSource` object representing the created output data.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxDataStep`, `rxImport`, `rxTransform`.

Examples

```
# rxFeaturize basically allows you to access data from the MicrosoftML transforms
# In this example we'll look at getting the output of the categorical transform

# Create the data
categoricalData <- data.frame(
  placesVisited = c(
    "London",
    "Brunei",
    "London",
    "Paris",
    "Seria"
  ),
  stringsAsFactors = FALSE
)

# Invoke the categorical transform
categorized <- rxFeaturize(
  data = categoricalData,
  mlTransforms = list(categorical(vars = c(xDataCat = "placesVisited")))
)

# Now let's look at the data
categorized
```

`rxHashEnv` *An environment object used to store package-wide state.*

Description

An environment object used to store package-wide state.

Usage

```
rxHashEnv
```

Format

An object of class `environment` of length 2.

`rxLogisticRegression` *Logistic Regression*

Description

Machine Learning Logistic Regression

Usage

```
rxLogisticRegression(formula = NULL, data, type = c("binary", "multiClass"),
  l2Weight = 1, l1Weight = 1, optTol = 1e-07, memorySize = 20,
  initWtsScale = 0, maxIterations = 2147483647, showTrainingStats = FALSE,
  sgdInitTol = 0, trainThreads = NULL, denseOptimizer = FALSE,
  normalize = "auto", mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula`	The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.
`data`	A data source object or a character string specifying a '`.xdf`' file or a data frame object.
`type`	A character string that specifies the type of Logistic Regression: `"binary"` for the default binary classification logistic regression or `"multi"` for multinomial logistic regression.
`l2Weight`	The L2 regularization weight. Its value must be greater than or equal to `0` and the default value is set to `1`.
`l1Weight`	The L1 regularization weight. Its value must be greater than or equal to `0` and the default value is set to `1`.

`optTol` Threshold value for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. Smaller values are slower, but more accurate. The default value is `1e-07`.

`memorySize` Memory size for L-BFGS, specifying the number of past positions and gradients to store for the computation of the next step. This optimization parameter limits the amount of memory that is used to compute the magnitude and direction of the next step. When you specify less memory, training is faster but less accurate. Must be greater than or equal to `1` and the default value is `20`.

`initWtsScale` Sets the initial weights diameter that specifies the range from which values are drawn for the initial weights. These weights are initialized randomly from within this range. For example, if the diameter is specified to be `d`, then the weights are uniformly distributed between `-d/2` and `d/2`. The default value is `0`, which specifies that allthe weights are initialized to `0`.

`maxIterations` Sets the maximum number of iterations. After this number of steps, the algorithm stops even if it has not satisfied convergence criteria.

`showTrainingStats`
Specify `TRUE` to show the statistics of training data and the trained model; otherwise, `FALSE`. The default value is `FALSE`. For additional information about model statistics, see `summary.mlModel`.

`sgdInitTol` Set to a number greater than 0 to use Stochastic Gradient Descent (SGD) to find the initial parameters. A non-zero value set specifies the tolerance SGD uses to determine convergence. The default value is `0` specifying that SGD is not used.

`trainThreads` The number of threads to use in training the model. This should be set to the number of cores on the machine. Note that L-BFGS multi-threading attempts to load dataset into memory. In case of out-of-memory issues, set `trainThreads` to `1` to turn off multi-threading. If `NULL` the number of threads to use is determined internally. The default value is `NULL`.

`denseOptimizer` If `TRUE`, forces densification of the internal optimization vectors. If `FALSE`, enables the logistic regression optimizer use sparse or dense internal states as it finds appropriate. Setting `denseOptimizer` to `TRUE` requires the internal optimizer to use a dense internal state, which may help alleviate load on the garbage collector for some varieties of larger problems.

`normalize` Specifies the type of automatic normalization used:

- `"auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"no"`: no normalization is performed.
- `"yes"`: normalization is performed.
- `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval [a, b] where `-1 <= a <= 0` and `0 <= b <= 1` and `b - a = 1`. This normalizer preserves sparsity by mapping zero to zero.

`mlTransforms` Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that aresupported. These transformations are performed after any specified R transformations. The default avlue is `NULL`.

`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.
`rowSelection`	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.
`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.
`transformFunc`	The variable transformation function. See rxTransform for details.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`ensemble`	Control parameters for ensembling.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

Logistic Regression is a classification method used to predict the value of a categorical dependent variable from its relationship to one or more independent variables assumed to have a logistic distribution. If the dependent variable has only two possible values (success/failure), then the logistic regression is binary. If the dependent variable has more than two possible values (blood type given diagnostic test results), then the logistic regression is multinomial.

The optimization technique used for `rxLogisticRegression` is the limited memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS). Both the L-BFGS and regular BFGS algorithms use quasi-Newtonian methods to estimate the computationally intensive Hessian matrix in the equation used by Newton's method to calculate steps. But the L-BFGS approximation uses only a limited amount of memory to compute the next step direction, so that it is especially suited for problems with a large number of variables. The `memorySize` parameter specifies the number of past positions and gradients to store for use in the computation of the next step.

This learner can use elastic net regularization: a linear combination of L1 (lasso) and L2 (ridge) regularizations. Regularization is a method that can render an ill-posed problem more tractable by imposing constraints that provide information to supplement the data and that prevents overfitting by penalizing models with extreme coefficient values. This can improve the generalization of the model learned by selecting the optimal complexity in the bias-variance tradeoff. Regularization works by adding the penalty that is associated with coefficient values to the error of the hypothesis. An accurate model with extreme coefficient values would be penalized more, but a less accurate model with more conservative values would be penalized less. L1 and L2 regularization have different effects and uses that are complementary in certain respects.

- `l1Weight`: can be applied to sparse models, when working with high-dimensional data. It pulls small weights associated features that are relatively unimportant towards 0.
- `l2Weight`: is preferable for data that is not sparse. It pulls large weights towards zero.

Adding the ridge penalty to the regularization overcomes some of lasso's limitations. It can improve its predictive accuracy, for example, when the number of predictors is greater than the sample size. If `x = l1Weight` and `y = l2Weight`, `ax + by = c` defines the linear span of the regularization terms. The default values of `x` and `y` are both `1`. An agressive regularization can harm predictive capacity by excluding important variables out of the model. So choosing the optimal values for the regularization parameters is important for the performance of the logistic regression model.

Value

- `rxLogisticRegression`: A `rxLogisticRegression` object with the trained model.
- `LogisticReg`: A learner specification object of class `maml` for the Logistic Reg trainer.

Note

This algorithm will attempt to load the entire dataset into memory when `trainThreads > 1` (multi-threading).

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: L-BFGS

Wikipedia: Logistic regression

Scalable Training of L1-Regularized Log-Linear Models

Test Run - L1 and L2 Regularization for Machine Learning

See Also

rxFastTrees, rxFastForest, rxFastLinear, rxNeuralNet, rxOneClassSvm, featurizeText, categorical, categoricalHash, rxPredict.mlModel.

Examples

```
# Estimate a logistic regression model
logitModel <- rxLogisticRegression(isCase ~ age + parity + education + spontaneous + induced,
                  transforms = list(isCase = case == 1),
                  data = infert)
# Print a summary of the model
summary(logitModel)

# Score to a data frame
scoreDF <- rxPredict(logitModel, data = infert,
    extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data = scoreDF)
plot(roc1)
rxAuc(roc1)

#########################################################################################
# Multi-class logistic regression
testObs <- rnorm(nrow(iris)) > 0
testIris <- iris[testObs,]
trainIris <- iris[!testObs,]
multiLogit <- rxLogisticRegression(
    formula = Species~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    type = "multiClass", data = trainIris)

# Score the model
scoreMultiDF <- rxPredict(multiLogit, data = testIris,
    extraVarsToWrite = "Species")
# Print the first rows of the data frame with scores
head(scoreMultiDF)
# Look at confusion matrix
table(scoreMultiDF$Species, scoreMultiDF$PredictedLabel)

# Look at the observations with incorrect predictions
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel
scoreMultiDF[badPrediction,]
```

---

rxNeuralNet *Neural Net*

Description

Neural networks for regression modeling and for Binary and multi-class classification.

Usage

```
rxNeuralNet(formula = NULL, data, type = c("binary", "multiClass",
  "regression"), numHiddenNodes = 100, numIterations = 100,
  optimizer = sgd(), netDefinition = NULL, initWtsDiameter = 0.1,
  maxNorm = 0, acceleration = c("sse", "gpu"), miniBatchSize = 1,
  normalize = "auto", mlTransforms = NULL, mlTransformVars = NULL,
  rowSelection = NULL, transforms = NULL, transformObjects = NULL,
  transformFunc = NULL, transformVars = NULL, transformPackages = NULL,
  transformEnvir = NULL, blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula` The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.

`data` A data source object or a character string specifying a '`.xdf`' file or a data frame object.

`type` A character string denoting Fast Tree type:

- `"binary"` for the default binary classification neural network.
- `"multiClass"` for multi-class classification neural network.
- `"regression"` for a regression neural network.

`numHiddenNodes` The default number of hidden nodes in the neural net. The default value is 100.

`numIterations` The number of iterations on the full training set. The default value is 100.

`optimizer` A list specifying either the `sgd` or `adaptive` optimization algorithm. This list can be created using `sgd` or `adaDeltaSgd`. The default value is `sgd`.

`netDefinition` The Net# definition of the structure of the neural network. For more information about the Net# language, see Reference Guide

`initWtsDiameter`

Sets the initial weights diameter that specifies the range from which values are drawn for the initial learning weights. The weights are initialized randomly from within this range. The default value is 0.1.

`maxNorm` Specifies an upper bound to constrain the norm of the incoming weight vector at each hidden unit. This can be very important in maxout neural networks as well as in cases where training produces unbounded weights.

`acceleration` Specifies the type of hardware acceleration to use. Possible values are "sse" and "gpu". For GPU acceleration, it is recommended to use a miniBatchSize greater than one. If you want to use the GPU acceleration, there are additional manual setup steps are required:

- Download and install NVidia CUDA Toolkit 6.5 (CUDA Toolkit).
- Download and install NVidia cuDNN v2 Library (cudnn Library).
- Find the libs directory of the MicrosoftRML package by calling `system.file("mxLibs/x64", p`
- Copy cublas64_65.dll, cudart64_65.dll and cusparse64_65.dll from the CUDA Toolkit 6.5 into the libs directory of the MicrosoftML package.
- Copy cudnn64_65.dll from the cuDNN v2 Library into the libs directory of the MicrosoftML package.

`miniBatchSize`	Sets the mini-batch size. Recommended values are between 1 and 256. This parameter is only used when the acceleration is GPU. Setting this parameter to a higher value improves the speed of training, but it might negatively affect the accuracy. The default value is 1.
`normalize`	Specifies the type of automatic normalization used: • `"auto"`: if normalization is needed, it is performed automatically. This is the default choice. • `"no"`: no normalization is performed. • `"yes"`: normalization is performed. • `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed. Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval [a, b] where `-1 <= a <= 0` and `0 <= b <= 1` and `b - a = 1`. This normalizer preserves sparsity by mapping zero to zero.
`mlTransforms`	Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.
`mlTransformVars`	Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.
`rowSelection`	Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.
`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.
`transformFunc`	The variable transformation function. See rxTransform for details.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments

	or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`ensemble`	Control parameters for ensembling.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

A neural network is a class of prediction models inspired by the human brain. A neural network can be represented as a weighted directed graph. Each node in the graph is called a neuron. The neurons in the graph are arranged in layers, where neurons in one layer are connected by a weighted edge (weights can be 0 or positive numbers) to neurons in the next layer. The first layer is called the input layer, and each neuron in the input layer corresponds to one of the features. The last layer of the function is called the output layer. So in the case of binary neural networks it contains two output neurons, one for each class, whose values are the probabilities of belonging to each class. The remaining layers are called hidden layers. The values of the neurons in the hidden layers and in the output layer are set by calculating the weighted sum of the values of the neurons in the previous layer and applying an activation function to that weighted sum. A neural network model is defined by the structure of its graph (namely, the number of hidden layers and the number of neurons in each hidden layer), the choice of activation function, and the weights on the graph edges. The neural network algorithm tries to learn the optimal weights on the edges based on the training data.

Although neural networks are widely known for use in deep learning and modeling complex problems such as image recognition, they are also easily adapted to regression problems. Any class of statistical models can be considered a neural network if they use adaptive weights and can approximate non-linear functions of their inputs. Neural network regression is especially suited to problems where a more traditional regression model cannot fit a solution.

Value

- `rxNeuralNet`: an `rxNeuralNet` object with the trained model.
- `NeuralNet`: a learner specification object of class `maml` for the Neural Net trainer.

Note

This algorithm is single-threaded and will not attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: Artificial neural network

See Also

rxFastTrees, rxFastForest, rxFastLinear, rxLogisticRegression, rxOneClassSvm, featurizeText, categorical, categoricalHash, rxPredict.mlModel.

Examples

```
# Estimate a binary neural net
rxNeuralNet1 <- rxNeuralNet(isCase ~ age + parity + education + spontaneous + induced,
                  transforms = list(isCase = case == 1),
                  data = infert)

# Score to a data frame
scoreDF <- rxPredict(rxNeuralNet1, data = infert,
    extraVarsToWrite = "isCase",
    outData = NULL) # return a data frame

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data = scoreDF)
plot(roc1)
rxAuc(roc1)

#########################################################################
# Regression neural net

# Create an xdf file with the attitude data
myXdf <- tempfile(pattern = "tempAttitude", fileext = ".xdf")
rxDataStep(attitude, myXdf, rowsPerRead = 50, overwrite = TRUE)
myXdfDS <- RxXdfData(file = myXdf)

attitudeForm <- rating ~ complaints + privileges + learning +
    raises + critical + advance

# Estimate a regression neural net
res2 <- rxNeuralNet(formula = attitudeForm,  data = myXdfDS,
    type = "regression")

# Score to data frame
scoreOut2 <- rxPredict(res2, data = myXdfDS,
    extraVarsToWrite = "rating")

# Plot the rating versus the score with a regression line
rxLinePlot(rating~Score, type = c("p","r"), data = scoreOut2)

# Clean up
```

```
file.remove(myXdf)

################################################################################
# Multi-class neural net
multiNN <- rxNeuralNet(
    formula = Species~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    type = "multiClass", data = iris)
scoreMultiDF <- rxPredict(multiNN, data = iris,
    extraVarsToWrite = "Species", outData = NULL)
# Print the first rows of the data frame with scores
head(scoreMultiDF)
# Compute % of incorrect predictions
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel
sum(badPrediction)*100/nrow(scoreMultiDF)
# Look at the observations with incorrect predictions
scoreMultiDF[badPrediction,]
```

`rxOneClassSvm` *OneClass SVM*

Description

Machine Learning One Class Support Vector Machines

Usage

```
rxOneClassSvm(formula = NULL, data, cacheSize = 100, kernel = rbfKernel(),
  epsilon = 0.001, nu = 0.1, shrink = TRUE, normalize = "auto",
  mlTransforms = NULL, mlTransformVars = NULL, rowSelection = NULL,
  transforms = NULL, transformObjects = NULL, transformFunc = NULL,
  transformVars = NULL, transformPackages = NULL, transformEnvir = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"),
  ensemble = ensembleControl(), ...)
```

Arguments

`formula`	The formula as described in `rxFormula`. Interaction terms and `F()` are not currently supported in the **MicrosoftML**.
`data`	A data source object or a character string specifying a '`.xdf`' file or a data frame object.
`cacheSize`	The maximal size in MB of the cache that stores the training data. Increase this for large training sets. The default value is 100 MB.
`kernel`	A character string representing the kernel used for computing inner products. For more information, see `maKernel`. The following choices are available: • `rbfKernel()`: Radial basis function kernel. It's parameter representsgamma in the term `exp(-gamma|x-y|^2`. If not specified, it defaults to 1 divided by the number of features used. For example, `rbfKernel(gamma = .1)`. This is the default value. • `linearKernel()`: Linear kernel.

- `polynomialKernel()`: Polynomial kernel with parameter names `a`, `bias`, and `deg` in the term `(a*<x,y> + bias)^deg`. The `bias`, defaults to `0`. The degree, `deg`, defaults to `3`. If `a` is not specified, it is set to `1` divided by the number of features. For example, `maKernelPoynomial(bias = 0, deg = 3)`.
- `sigmoidKernel()`: Sigmoid kernel with parameter names `gamma` and `coef0` in the term `tanh(gamma*<x,y> + coef0)`. `gamma`, defaults to to `1` divided by the number of features. The parameter `coef0` defaults to `0`. For example, `sigmoidKernel(gamma = .1, coef0 = 0)`.

`epsilon` The threshold for optimizer convergence. If the improvement between iterations is less than the threshold, the algorithm stops and returns the current model. The value must be greater than or equal to `.Machine$double.eps`. The default value is 0.001.

`nu` The trade-off between the fraction of outliers and the number of support vectors (represented by the Greek letter nu). Must be between 0 and 1, typically between 0.1 and 0.5. The default value is 0.1.

`shrink` Uses the shrinking heuristic if `TRUE`. In this case, some samples will be "shrunk" during the training procedure, which may speed up training. The default value is `TRUE`.

`normalize` Specifies the type of automatic normalization used:

- `"auto"`: if normalization is needed, it is performed automatically. This is the default choice.
- `"no"`: no normalization is performed.
- `"yes"`: normalization is performed.
- `"warn"`: if normalization is needed, a warning message is displayed, but normalization is not performed.

Normalization rescales disparate data ranges to a standard scale. Feature scaling insures the distances between data points are proportional and enables various optimization methods such as gradient descent to converge much faster. If normalization is performed, a `MaxMin` normalizer is used. It normalizes values in an interval [a, b] where `-1 <= a <= 0` and `0 <= b <= 1` and `b - a = 1`. This normalizer preserves sparsity by mapping zero to zero.

`mlTransforms` Specifies a list of MicrosoftML transforms to be performed on the data before training or `NULL` if no transforms are to be performed. See `featurizeText`, `categorical`, and `categoricalHash`, for transformations that are supported. These transformations are performed after any specified R transformations. The default value is `NULL`.

`mlTransformVars`

Specifies a character vector of variable names to be used in `mlTransforms` or `NULL` if none are to be used. The default value is `NULL`.

`rowSelection` Specifies the rows (observations) from the data set that are to be used by the model with the name of a logical variable from the data set (in quotes) or with a logical expression using variables in the data set. For example, `rowSelection = "old"` will only use observations in which the value of the variable `old` is `TRUE`. `rowSelection = (age > 20` only uses observations in which the value of the `age` variable is between 20 and 65 and the value of the `log` of the `income` variable is greater than 10. The row selection is performed after processing any data transformations (see the arguments `transforms` or `transformFunc`). As with all expressions, `rowSelection` can be defined outside of the function call using the `expression` function.

`transforms`	An expression of the form `list(name = expression, ...)` that represents the first round of variable transformations. As with all expressions, `transforms` (or `rowSelection`) can be defined outside of the function call using the `expression` function.
`transformObjects`	A named list that contains objects that can be referenced by `transforms`, `transformsFunc`, and `rowSelection`.
`transformFunc`	The variable transformation function. See rxTransform for details.
`transformVars`	A character vector of input data set variables needed for the transformation function. See rxTransform for details.
`transformPackages`	A character vector specifying additional R packages (outside of those specified in `rxGetOption("transformPackages")`) to be made available and preloaded for use in variable transformation functions. For exmple, those explicitly defined in **RevoScaleR** functions via their `transforms` and `transformFunc` arguments or those defined implicitly via their `formula` or `rowSelection` arguments. The `transformPackages` argument may also be `NULL`, indicating that no packages outside `rxGetOption("transformPackages")` are preloaded.
`transformEnvir`	A user-defined environment to serve as a parent to all environments developed internally and used for variable data transformation. If `transformEnvir = NULL`, a new "hash" environment with parent `baseenv()` is used instead.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from `1` to `4` provide increasing amounts of information.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`ensemble`	Control parameters for ensembling.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

detection is to identify outliers that do not belong to some target class. This type of SVM is one-class because the training set contains only examples from the target class. It infers what properties are normal for the objects in the target class and from these properties predicts which examples are unlike the normal examples. This is useful for anomaly detection because the scarcity of training examples is the defining character of anomalies: typically there are very few examples of network intrusion, fraud, or other types of anomalous behavior.

Value

- `rxOneClassSvm`: A `rxOneClassSvm` object with the trained model.
- `OneClassSvm`: A learner specification object of class `maml` for the OneClass Svm trainer.

Note

This algorithm is single-threaded and will always attempt to load the entire dataset into memory.

Author(s)

Microsoft Corporation Microsoft Technical Support

References

Wikipedia: Anomaly detection

Microsoft Azure Machine Learning Studio: One-Class Support Vector Machine

Estimating the Support of a High-Dimensional Distribution

New Support Vector Algorithms

LIBSVM: A Library for Support Vector Machines

See Also

`rbfKernel`, `linearKernel`, `polynomialKernel`, `sigmoidKernel` `rxFastTrees`, `rxFastForest`, `rxFastLinear`, `rxLogisticRegression`, `rxNeuralNet`, `featurizeText`, `categorical`, `categoricalHash`, `rxPredict.mlModel`.

Examples

```
# Estimate a One-Class SVM model
trainRows <- c(1:30, 51:80, 101:130)
testRows = !(1:150 %in% trainRows)
trainIris <- iris[trainRows,]
testIris <- iris[testRows,]

svmModel <- rxOneClassSvm(
    formula = ~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    data = trainIris)

# Add additional non-iris data to the test data set
testIris$isIris <- 1
notIris <- data.frame(
    Sepal.Length = c(2.5, 2.6),
    Sepal.Width = c(.75, .9),
    Petal.Length = c(2.5, 2.5),
    Petal.Width = c(.8, .7),
    Species = c("not iris", "not iris"),
    isIris = 0)
testIris <- rbind(testIris, notIris)

scoreDF <- rxPredict(svmModel,
     data = testIris, extraVarsToWrite = "isIris")

# Look at the last few observations
tail(scoreDF)
# Look at average scores conditioned by 'isIris'
rxCube(Score ~ F(isIris), data = scoreDF)
```

`rxPredict.mlModel` *Score using a Microsoft R Machine Learning model*

Description

Reports per-instance scoring results in a data frame or RevoScaleR data source using a trained Microsoft R Machine Learning model with a RevoScaleR data source.

Usage

```
## S3 method for class 'mlModel'
rxPredict(modelObject, data, outData = NULL,
  writeModelVars = FALSE, extraVarsToWrite = NULL, suffix = NULL,
  overwrite = FALSE, dataThreads = NULL,
  blocksPerRead = rxGetOption("blocksPerRead"),
  reportProgress = rxGetOption("reportProgress"), verbose = 1,
  computeContext = rxGetOption("computeContext"), ...)
```

Arguments

`modelObject`	A model information object returned from a MicrosoftML model. For example, an object returned from `rxFastTrees` or `rxLogisticRegression`.
`data`	A **RevoScaleR** data source object, a data frame, or the path to a `.xdf` file.
`outData`	Output text or xdf file name or an `RxDataSource` with write capabilities in which to store predictions. If `NULL`, a data frame is returned. The default value is `NULL`.
`writeModelVars`	If `TRUE`, variables in the model are written to the output data set in addition to the scoring variables. If variables from the input data set are transformed in the model, the transformed variables are also included. The default value is `FALSE`.
`extraVarsToWrite`	`NULL` or character vector of additional variables names from the input data to include in the `outData`. If `writeModelVars` is `TRUE`, model variables are included as well. The default value is `NULL`.
`suffix`	A character string specifying suffix to append to the created scoring variable(s) or `NULL` in there is no suffix. The default value is `NULL`.
`overwrite`	If `TRUE`, an existing `outData` is overwritten; if `FALSE` an existing `outData` is not overwritten. The default value is `FALSE`.
`dataThreads`	An integer specifying the desired degree of parallelism in the data pipeline. If `NULL`, the number of threads used is determined internally. The default value is `NULL`.
`blocksPerRead`	Specifies the number of blocks to read for each chunk of data read from the data source.
`reportProgress`	An integer value that specifies the level of reporting on the row processing progress: • `0`: no progress is reported. • `1`: the number of processed rows is printed and updated. • `2`: rows processed and timings are reported. • `3`: rows processed and all timings are reported.

	The default value is 1.
`verbose`	An integer value that specifies the amount of output wanted. If `0`, no verbose output is printed during calculations. Integer values from 1 to 4 provide increasing amounts of information. The default value is 1.
`computeContext`	Sets the context in which computations are executed, specified with a valid `RxComputeContext`. Currently local and `RxInSqlServer` compute contexts are supported.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The following items are reported in the output by default: scoring on three variables for the binary classifiers: PredictedLabel, Score, and Probability; the Score for oneClassSvm and regression classifiers; PredictedLabel for Multi-class classifiers, plus a variable for each category prepended by the Score.

Value

A data frame or an `RxDataSource` object representing the created output data. By default, output from scoring binary classifiers include three variables: `PredictedLabel`, `Score`, and `Probability`; `rxOneClassSvm` and regression include one variable: `Score`; and multi-class classifiers include `PredictedLabel` plus a variable for each category prepended by `Score`. If a `suffix` is provided, it is added to the end of these output variable names.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastTrees`, `rxFastForest`, `rxLogisticRegression`, `rxNeuralNet`, `rxOneClassSvm`.

Examples

```
# Estimate a logistic regression model
infert1 <- infert
infert1$isCase <- (infert1$case == 1)
myModelInfo <- rxLogisticRegression(formula = isCase ~ age + parity + education + spontaneous + induced,
                       data = infert1)

# Create an xdf file with per-instance results using rxPredict
xdfOut <- tempfile(pattern = "scoreOut", fileext = ".xdf")
scoreDS <- rxPredict(myModelInfo, data = infert1,
    outData = xdfOut, overwrite = TRUE,
    extraVarsToWrite = c("isCase", "Probability"))

# Summarize results with an ROC curve
rxRocCurve(actualVarName = "isCase", predVarNames = "Probability", data = scoreDS)

# Use the built-in data set 'airquality' to create test and train data
DF <- airquality[!is.na(airquality$Ozone), ]
DF$Ozone <- as.numeric(DF$Ozone)
set.seed(12)
```

```
randomSplit <- rnorm(nrow(DF))
trainAir <- DF[randomSplit >= 0,]
testAir <- DF[randomSplit < 0,]
airFormula <- Ozone ~ Solar.R + Wind + Temp

# Regression Fast Tree for train data
fastTreeReg <- rxFastTrees(airFormula, type = "regression",
    data = trainAir)

# Put score and model variables in data frame, including the model variables
# Add the suffix "Pred" to the new variable
fastTreeScoreDF <- rxPredict(fastTreeReg, data = testAir,
    writeModelVars = TRUE, suffix = "Pred")

rxGetVarInfo(fastTreeScoreDF)

# Clean-up
file.remove(xdfOut)
```

`selectColumns` *Selects a set of columns, dropping all others*

Description

Selects a set of columns to retrain, dropping all others.

Usage

```
selectColumns(vars, ...)
```

Arguments

`vars`	Specifiies character vector or list of the names of the variables to keep.
`...`	Additional arguments sent to compute engine.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

selectFeatures *Machine Learning Feature Selection Transform*

Description

The feature selection transform selects features from the specified variables using the specified mode.

Usage

```
selectFeatures(vars, mode, ...)
```

Arguments

`vars`	A formula or a vector/list of strings specifying the name of variables upon which the feature selection is performed, if the mode is minCount(). For example, `~ var1 + var2 + var3`. If mode is mutualInformation(), a formula or a named list of strings describing the dependent variable and the independent variables. For example, `label ~ var1 + var2 + var3`.
`mode`	Specifies the mode of feature selection. This can be either `minCount` or `mutualInformation`.
`...`	Additional arguments to be passed directly to the Microsoft Compute Engine.

Details

The feature selection transform selects features from the specified variables using one of the two modes: count or mutual information. For more information, see `minCount` and `mutualInformation`.

Value

A `maml` object defining the transform.

See Also

`minCount` `mutualInformation`

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )

    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)

# Use a categorical hash transform which generated 128 features.
outModel1 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
   mlTransforms = list(categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7)))
summary(outModel1)

# Apply a categorical hash transform and a count feature selection transform
# which selects only those hash slots that has value.
outModel2 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures("reviewCatHash", mode = minCount())))
summary(outModel2)

# Apply a categorical hash transform and a mutual information feature selection transform
# which selects only 10 features with largest mutual information with the label.
outModel3 <- rxLogisticRegression(like~reviewCatHash, data = trainReviews, l1Weight = 0,
    mlTransforms = list(
categoricalHash(vars = c(reviewCatHash = "review"), hashBits = 7),
selectFeatures(like ~ reviewCatHash, mode = mutualInformation(numFeaturesToKeep = 10))))
summary(outModel3)
```

`stopwordsDefault` *Machine Learning Text Transform*

Description

Text transforms that can be performed on data before training a model.

Usage

```
stopwordsDefault()

stopwordsCustom(dataFile = "")

termDictionary(terms = "", dataFile = "", sort = "occurrence")

featurizeText(vars, language = "English", stopwordsRemover = NULL,
  case = "lower", keepDiacritics = FALSE, keepPunctuations = TRUE,
  keepNumbers = TRUE, dictionary = NULL,
  wordFeatureExtractor = ngramCount(), charFeatureExtractor = NULL,
  vectorNormalizer = "l2", ...)
```

Arguments

`dataFile` character: <string>. Data file containing the terms (short form data).

`terms` An optional character vector of terms or categories.

`sort` Specifies how to order items when vectorized. Two orderings are supported:

- `"occurrence"`: items appear in the order encountered.
- `"value"`: items are sorted according to their default comparison. For example, text sorting will be case sensitive (e.g., 'A' then 'Z' then 'a').

`vars` A named list of character vectors of input variable names and the name of the output variable. Note that the input variables must be of the same type. For one-to-one mappings between input and output variables, a named character vector can be used.

`language` Secifies the language used in the data set. The following values are supported:

- `"AutoDetect"`: for automatic language detection.
- `"English"`.
- `"French"`.
- `"German"`.
- `"Dutch"`.
- `"Italian"`.
- `"Spanish"`.
- `"Japanese"`.

`stopwordsRemover`

Specifies the stopwords remover to use. There are three options supported:

- `NULL` No stopwords remover is used.
- `stopwordsDefault`: A precompiled language-specific lists of stop words is used that includes the most common words from Microsoft Office.
- `stopwordsCustom`: A user-defined list of stopwords. It accepts the following option: `dataFile`.

The default value is `NULL`.

`case` Text casing using the rules of the invariant culture. Takes the following values:

- `"lower"`.
- `"upper"`.
- `"none"`.

The default value is `"lower"`.

`keepDiacritics` `FALSE` to remove diacritical marks; `TRUE` to retain diacritical marks. The default value is `FALSE`.

`keepPunctuations` `FALSE` to remove punctuation; `TRUE` to retain punctuation. The default value is `TRUE`.

`keepNumbers` `FALSE` to remove numbers; `TRUE` to retain numbers. The default value is `TRUE`.

`dictionary` A `termDictionary` of whitelisted terms which accepts the following options:

- `terms`,
- `dataFile`, and
- `sort`.

The default value is `NULL`. Note that the stopwords list takes precedence over the dictionary whitelist as the stopwords are removed before the dictionary terms are whitelisted.

`wordFeatureExtractor` Specifies the word feature extraction arguments. There are two different feature extraction mechanisms:

- `ngramCount`: Count-based feature extraction (equivalent to WordBag). It accepts the following options: `maxNumTerms` and `weighting`.
- `ngramHash`: Hashing-based feature extraction (equivalent to WordHashBag). It accepts the following options: `hashBits`, `seed`, `ordered` and `invertHash`.

The default value is `ngramCount`.

`charFeatureExtractor` Specifies the char feature extraction arguments. There are two different feature extraction mechanisms:

- `ngramCount`: Count-based feature extraction (equivalent to WordBag). It accepts the following options: `maxNumTerms` and `weighting`.
- `ngramHash`: Hashing-based feature extraction (equivalent to WordHashBag). It accepts the following options: `hashBits`, `seed`, `ordered` and `invertHash`.

The default value is `NULL`.

`vectorNormalizer` Normalize vectors (rows) individually by rescaling them to unit norm. Takes one of the following values:

- `"none"`.
- `"l2"`.
- `"l1"`.
- `"linf"`.

The default value is `"l2"`.

`...` Additional arguments sent to the compute engine.

Details

The `featurizeText` transform produces a bag of counts of sequences of consecutive words, called n-grams, from a given corpus of text. There are two ways it can do this:

- build a dictionary of n-grams and use the id in the dictionary as the index in the bag;
- hash each n-gram and use the hash value as the index in the bag.

The purpose of hashing is to convert variable-length text documents into equal-length numeric feature vectors, to support dimensionality reduction and to make the lookup of feature weights faster.

The text transform is applied to text input columns. It offers language detection, tokenization, stopwords removing, text normalization and feature generation. It supports the following languages by default: English, French, German, Dutch, Italian, Spanish and Japanese.

The n-grams are represented as count vectors, with vector slots corresponding either to n-grams (created using `ngramCount`) or to their hashes (created using `ngramHash`). Embedding ngrams in a vector space allows their contents to be compared in an efficient manner. The slot values in the vector can be weighted by the following factors:

- term frequency - The number of occurrences of the slot in the text
- inverse document frequency - A ratio (the logarithm of inverse relative slot frequency) that measures the information a slot provides by determining how common or rare it is across the entire text.
- term frequency-inverse document frequency - the product term frequency and the inverse document frequency.

Value

A `maml` object defining the transform.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`ngramCount`, `ngramHash`, `rxFastTrees`, `rxFastForest`, `rxNeuralNet`, `rxOneClassSvm`, `rxLogisticRegression`.

Examples

```
trainReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Do not like it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I kind of hate it",
        "I do like it",
        "I really hate it",
        "It is very good",
        "I hate it a bunch",
        "I love it a bunch",
        "I hate it",
        "I like it very much",
        "I hate it very much.",
        "I really do love it",
        "I really do hate it",
        "Love it!",
        "Hate it!",
        "I love it",
        "I hate it",
        "I love it",
        "I hate it",
        "I love it"),
     like = c(TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE, FALSE, TRUE, FALSE, TRUE,
        FALSE, TRUE, FALSE, TRUE), stringsAsFactors = FALSE
    )

    testReviews <- data.frame(review = c(
        "This is great",
        "I hate it",
        "Love it",
        "Really like it",
        "I hate it",
        "I like it a lot",
        "I love it",
        "I do like it",
        "I really hate it",
        "I love it"), stringsAsFactors = FALSE)


outModel <- rxLogisticRegression(like ~ reviewTran, data = trainReviews,
    mlTransforms = list(featurizeText(vars = c(reviewTran = "review"),
    stopwordsRemover = stopwordsDefault(), keepPunctuations = FALSE)))
# 'hate' and 'love' have non-zero weights
summary(outModel)

# Use the model to score
scoreOutDF5 <- rxPredict(outModel, data = testReviews,
    extraVarsToWrite = "review")
scoreOutDF5
```

`summary.mlModel` *Summary of a Microsoft R Machine Learning model.*

Description

Summary of a Microsoft R Machine Learning model.

Usage

```
## S3 method for class 'mlModel'
summary(object, top = 20, ...)
```

Arguments

`object`	A model object returned from a **MicrosoftML** analysis.
`top`	Specifies the count of top coefficients to show in the summary for linear models such as `rxLogisticRegression` and `rxFastLinear`. The bias appears first, followed by other weights, sorted by their absolute values in descending order. If set to `NULL`, all non-zero coefficients are shown. Otherwise, only the first `top` coefficients are shown.
`...`	Additional arguments to be passed to the summary method.

Details

Provides summary information about the original function call, the data set used to train the model, and statistics for coefficients in the model.

Value

The `summary` method of the **MicrosoftML** analysis objects returns a list that includes the original function call and the underlying parameters used. The `coef` method returns a named vector of weights, processing information from the model object.

For `rxLogisticRegression`, the following statistics may also present in the summary when `showTrainingStats` is set to `TRUE`.

`training.size`	The size, in terms of row count, of the data set used to train the model.
`deviance`	The model deviance is given by `-2 * ln(L)` where `L` is the likelihood of obtaining the observations with all features incorporated in the model.
`null.deviance`	The null deviance is given by `-2 * ln(L0)` where `L0` is the likelihood of obtaining the observations with no effect from the features. The null model includes the bias if there is one in the model.
`aic`	The AIC (Akaike Information Criterion) is defined as `2 * k + deviance`, where `k` is the number of coefficients of the model. The bias counts as one of the coefficients. The AIC is a measure of the relative quality of the model. It deals with the trade-off between the goodness of fit of the model (measured by deviance) and the complexity of the model (measured by number of coefficients).

`coefficients.stats`

This is a data frame containing the statistics for each coefficient in the model. For each coefficient, the following statistics are shown. The bias appears in the first row, and the remaining coefficients in the ascending order of p-value.

- EstimateThe estimated coefficient value of the model.
- Std ErrorThis is the square root of the large-sample variance of the estimate of the coefficient.
- z-ScoreWe can test against the null hypothesis, which states that the coefficient should be zero, concerning the significance of the coefficient by calculating the ratio of its estimate and its standard error. Under the null hypothesis, if there is no regularization applied, the estimate of the concerning coefficient follows a normal distribution with mean 0 and a standard deviation equal to the standard error computed above. The z-score outputs the ratio between the estimate of a coefficient and the standard error of the coefficient.
- Pr(>|z|)This is the corresponding p-value for the two-sided test of the z-score. Based on the significance level, a significance indicator is appended to the p-value. If `F(x)` is the CDF of the standard normal distribution `N(0, 1)`, then `P(>|z|) = 2 -2 * F(|z|)`.

Author(s)

Microsoft Corporation Microsoft Technical Support

See Also

`rxFastTrees`, `rxFastForest`, `rxFastLinear`, `rxOneClassSvm`, `rxNeuralNet`, `rxLogisticRegression`.

Examples

```
# Estimate a logistic regression model
logitModel <- rxLogisticRegression(isCase ~ age + parity + education + spontaneous + induced,
                  transforms = list(isCase = case == 1),
                  data = infert)
# Print a summary of the model
summary(logitModel)

# Score to a data frame
scoreDF <- rxPredict(logitModel, data = infert,
    extraVarsToWrite = "isCase")

# Compute and plot the Radio Operator Curve and AUC
roc1 <- rxRoc(actualVarName = "isCase", predVarNames = "Probability", data = scoreDF)
plot(roc1)
rxAuc(roc1)

#########################################################################################
# Multi-class logistic regression
testObs <- rnorm(nrow(iris)) > 0
testIris <- iris[testObs,]
trainIris <- iris[!testObs,]
multiLogit <- rxLogisticRegression(
    formula = Species~Sepal.Length + Sepal.Width + Petal.Length + Petal.Width,
    type = "multiClass", data = trainIris)

# Score the model
scoreMultiDF <- rxPredict(multiLogit, data = testIris,
    extraVarsToWrite = "Species")
# Print the first rows of the data frame with scores
head(scoreMultiDF)
# Look at confusion matrix
table(scoreMultiDF$Species, scoreMultiDF$PredictedLabel)

# Look at the observations with incorrect predictions
badPrediction = scoreMultiDF$Species != scoreMultiDF$PredictedLabel
scoreMultiDF[badPrediction,]
```

`tlcBinaryNeuralNetwork` *BinaryClassifierTrainer, Trainer: 'BinaryNeuralNetwork'*

Description

A binary neural network is a binary classification algorithm that uses neural network with two outputs to predict a certain binary value. TLC supports various types of neural networks, including deep neural networks (DNNs) and convolutional neural networks (CNN) via Net# language.

Usage

```
tlcBinaryNeuralNetwork(lossFunction = "CrossEntropy",
  defaultHiddenNodes = 100, netFileName = "", numIterations = 100,
  displayRefresh = 1, optimizationAlgorithm = "sgd",
  initWtsDiameter = 0.1, maxNorm = 0, earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, pruning = FALSE, pruningFactor = 0.01,
  pruningRounds = 10, pruningRoundIterations = 5, acceleration = "avx",
  preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
  miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
  hiddenDropoutRate = 0, netDefinition = "")
```

Arguments

`lossFunction`	list: <name><options>. Loss function Default value:'CrossEntropy' (short form loss)
`defaultHiddenNodes`	integer: <int>. Default number of hidden nodes Default value:'100' (short form hidden)
`netFileName`	character: <string>. Net file name (short form filename)
`numIterations`	integer: <int>. Number of training iterations Default value:'100' (short form iter)
`displayRefresh`	integer: <int>. Display refresh frequency in number iterations Default value:'1' (short form refresh)
`optimizationAlgorithm`	list: <name><options>. Optimization algorithm (Adadelta or SGD) Default value:'sgd' (short form algo)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0.1' (short form initwts)
`maxNorm`	double: <float>. Constrains the norm of incoming weights of a node Default value:'0'
`earlyStoppingRule`	list: <name><options>. Early stopping rule (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics Default value:'0' (short form esmt)
`pruning`	logical: [+|-]. Enable post-training pruning (Optimal Brain Damage) Default value:'-' (short form prune)
`pruningFactor`	double: <float>. Pruning factor: % of weights removed each pruning iteration Default value:'0.01' (short form prunefact)
`pruningRounds`	integer: <int>. Number of pruning rounds Default value:'10' (short form pruneround)
`pruningRoundIterations`	integer: <int>. Number of pruning round iterations Default value:'5' (short form pruneiter)
`acceleration`	list: <name><options>. Hardware acceleration level Default value:'avx' (short form accel)
`preTrainerType`	character: [NoPreTrainer|Greedy]. Net Pre-Trainer Default value:'NoPreTrainer' (short form pretrain)
`preTrainingEpoch`	integer: <int>. Number of epochs for pre-training. If not set, defaults to numIterations(iter). (short form prepoch)
`miniBatchSize`	integer: <int>. Mini-batch size Default value:'1' (short form mbsize)
`shuffle`	logical: [+|-]. Whether to shuffle for each training iteration Default value:'+' (short form shuf)

`inputDropoutRate`	double: <float>. Input dropout rate Default value:'0' (short form idrop)
`hiddenDropoutRate`	double: <float>. Hidden dropout rate Default value:'0' (short form hdrop)
`netDefinition`	character: <string>. Neural network definition (short form net)
`...`	: . hidden arguments

Value

a character string defining: BinaryNeuralNetwork (BinaryClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lossFunction = "CrossEntropy", defaultHiddenNodes = 100,
    netFileName = "", numIterations = 100, displayRefresh = 1,
    optimizationAlgorithm = "sgd", initWtsDiameter = 0.1, maxNorm = 0,
    earlyStoppingRule = list(), earlyStoppingMetrics = 0, pruning = FALSE,
    pruningFactor = 0.01, pruningRounds = 10, pruningRoundIterations = 5,
    acceleration = "avx", preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
    miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
    hiddenDropoutRate = 0, netDefinition = "", ...)
{
    params <- character()
    params <- mluPasteArg(lossFunction, "list", params)
    params <- mluPasteArg(defaultHiddenNodes, "integer", params)
    params <- mluPasteArg(netFileName, "character", params)
    params <- mluPasteArg(numIterations, "integer", params)
    params <- mluPasteArg(displayRefresh, "integer", params)
    params <- mluPasteArg(optimizationAlgorithm, "list", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(maxNorm, "double", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(pruning, "logical", params)
    params <- mluPasteArg(pruningFactor, "double", params)
    params <- mluPasteArg(pruningRounds, "integer", params)
    params <- mluPasteArg(pruningRoundIterations, "integer", params)
    params <- mluPasteArg(acceleration, "list", params)
    params <- mluPasteArg(preTrainerType, "character", params)
    params <- mluPasteArg(preTrainingEpoch, "integer", params)
    params <- mluPasteArg(miniBatchSize, "integer", params)
    params <- mluPasteArg(shuffle, "logical", params)
    params <- mluPasteArg(inputDropoutRate, "double", params)
    params <- mluPasteArg(hiddenDropoutRate, "double", params)
    params <- mluPasteArg(netDefinition, "character", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("BinaryNeuralNetwork{%s}", params)
    return(structure(params, class = c("BinaryNeuralNetwork",
        "BinaryClassifierTrainer", "maml", "character")))
}
```

tlcFastForestClassification *BinaryClassifierTrainer, Trainer: 'FastForestClassification'*

Description

Uses a random forest learner to perform binary classification.

Usage

```
tlcFastForestClassification(featureFraction = 0.7,
  baggingTrainFraction = 0.7, splitFraction = 0.7, numThreads = 8,
  rngSeed = 123, entropyCoefficient = 0, histogramPoolSize = -1,
  diskTranspose = FALSE, maxBins = 255, sparsifyThreshold = 0.7,
  featureFirstUsePenalty = 0, featureReusePenalty = 0,
  gainConfidenceLevel = 0, softmaxTemperature = 0, executionTimes = FALSE,
  numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100,
  smoothing = 0, testFrequency = 2147483647, baggingSize = 1)
```

Arguments

`featureFraction`	double: <float>. The fraction of features (chosen randomly) to use on each iteration Default value:'0.7' (short form ff)
`baggingTrainFraction`	double: <float>. The fraction of Instances (chosen randomly) to use on each iteration Default value:'0.7' (short form bagfrac)
`splitFraction`	double: <float>. The fraction of features (chosen randomly) to use on each split Default value:'0.7' (short form sf)
`numThreads`	integer: <int>. The number of threads to use Default value:'8' (short form t)
`rngSeed`	integer: <int>. The seed of the random number generator Default value:'123' (short form r1)
`entropyCoefficient`	double: <float>. The entropy (regularization) coefficient between 0 and 1 Default value:'0' (short form e)
`histogramPoolSize`	integer: <int>. The number of histograms in the pool (between 2 and numLeaves) Default value:'-1' (short form ps)

`diskTranspose`	logical: [+|-]. Whether to utilize the disk when performing the transpose Default value:'-' (short form dt)
`maxBins`	integer: <int>. Maximum number of distinct values (bins) per feature Default value:'255' (short form mb)
`sparsifyThreshold`	double: <float>. Sparsity level needed to use sparse feature representation Default value:'0.7' (short form sp)
`featureFirstUsePenalty`	double: <float>. The feature first use penalty coefficient Default value:'0' (short form ffup)
`featureReusePenalty`	double: <float>. The feature re-use penalty (regularization) coefficient Default value:'0' (short form frup)
`gainConfidenceLevel`	double: <float>. Tree fitting gain confidence requirement (should be in the range [0,1)). Default value:'0' (short form gainconf)
`softmaxTemperature`	double: <float>. The temperature of the randomized softmax distribution for choosing the feature Default value:'0' (short form smtemp)
`executionTimes`	logical: [+|-]. Print execution time breakdown to stdout Default value:'-' (short form et)
`numLeaves`	integer: <int>. The max number of leaves in each regression tree Default value:'20' (short form nl)
`minDocumentsInLeafs`	integer: <int>. The minimal number of documents allowed in a leaf of a regression tree, out of the subsampled data Default value:'10' (short form mil)
`numTrees`	integer: <int>. Number of weak hypotheses in the ensemble Default value:'100' (short form iter)
`smoothing`	double: <float>. Smoothing paramter for tree regularization Default value: '0.0' (short form s)
`testFrequency`	integer: <int>. Calculate metric values for train/valid/test every k rounds Default value: '2147483647' (short form tf)
`baggingSize`	integer: <int>. Number of trees in each bag (0 for disabling bagging) Default value: '0' (short form bag)
`...`	: . hidden arguments

Value

a character string defining: FastForestClassification (BinaryClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (featureFraction = 0.7, baggingTrainFraction = 0.7,
    splitFraction = 0.7, numThreads = 8, rngSeed = 123,
    entropyCoefficient = 0, histogramPoolSize = -1, diskTranspose = FALSE,
    maxBins = 255, sparsifyThreshold = 0.7, featureFirstUsePenalty = 0,
    featureReusePenalty = 0, gainConfidenceLevel = 0, softmaxTemperature = 0,
    executionTimes = FALSE, numLeaves = 20, minDocumentsInLeafs = 10,
    numTrees = 100, ...)
{
    params <- character()
    params <- mluPasteArg(featureFraction, 'double', params)
    params <- mluPasteArg(baggingTrainFraction, 'double', params)
    params <- mluPasteArg(splitFraction, 'double', params)
    params <- mluPasteArg(numThreads, 'integer', params)
    params <- mluPasteArg(rngSeed, 'integer', params)
    params <- mluPasteArg(entropyCoefficient, 'double', params)
    params <- mluPasteArg(histogramPoolSize, 'integer', params)
    params <- mluPasteArg(diskTranspose, 'logical', params)
    params <- mluPasteArg(maxBins, 'integer', params)
    params <- mluPasteArg(sparsifyThreshold, 'double', params)
    params <- mluPasteArg(featureFirstUsePenalty, 'double', params)
    params <- mluPasteArg(featureReusePenalty, 'double', params)
    params <- mluPasteArg(gainConfidenceLevel, 'double', params)
    params <- mluPasteArg(softmaxTemperature, 'double', params)
    params <- mluPasteArg(executionTimes, 'logical', params)
    params <- mluPasteArg(numLeaves, 'integer', params)
    params <- mluPasteArg(minDocumentsInLeafs, 'integer', params)
    params <- mluPasteArg(numTrees, 'integer', params)
    params <- mluPasteArg(smoothing, 'double', params)
    params <- mluPasteArg(testFrequency, 'integer', params)
    params <- mluPasteArg(baggingSize, 'integer', params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("FastForestClassification{%s}", params)
    return(structure(params, class = c("FastForestClassification",
        "BinaryClassifierTrainer", "maml", "character")))
  }
```

`tlcFastForestRegression` *RegressorTrainer, Trainer: 'FastForestRegression'*

Description

Trains a random forest to fit target values using least-squares.

Usage

```
tlcFastForestRegression(quantileSampleCount = 100, featureFraction = 0.7,
  baggingTrainFraction = 0.7, splitFraction = 0.7, numThreads = 8,
  rngSeed = 123, entropyCoefficient = 0, histogramPoolSize = -1,
  diskTranspose = FALSE, maxBins = 255, sparsifyThreshold = 0.7,
  featureFirstUsePenalty = 0, featureReusePenalty = 0,
  gainConfidenceLevel = 0, softmaxTemperature = 0, executionTimes = FALSE,
  numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100,
  smoothing = 0, testFrequency = 2147483647, baggingSize = 1)
```

Arguments

`quantileSampleCount`	integer: <int>. Number of labels to be sampled from each leaf to make the distribtuion Default value:'100' (short form qsc)
`featureFraction`	double: <float>. The fraction of features (chosen randomly) to use on each iteration Default value:'0.7' (short form ff)
`baggingTrainFraction`	double: <float>. The fraction of Instances (chosen randomly) to use on each iteration Default value:'0.7' (short form bagfrac)
`splitFraction`	double: <float>. The fraction of features (chosen randomly) to use on each split Default value:'0.7' (short form sf)
`numThreads`	integer: <int>. The number of threads to use Default value:'8' (short form t)
`rngSeed`	integer: <int>. The seed of the random number generator Default value:'123' (short form r1)
`entropyCoefficient`	double: <float>. The entropy (regularization) coefficient between 0 and 1 Default value:'0' (short form e)
`histogramPoolSize`	integer: <int>. The number of histograms in the pool (between 2 and numLeaves) Default value:'-1' (short form ps)
`diskTranspose`	logical: [+|-]. Whether to utilize the disk when performing the transpose Default value:'-' (short form dt)
`maxBins`	integer: <int>. Maximum number of distinct values (bins) per feature Default value:'255' (short form mb)
`sparsifyThreshold`	double: <float>. Sparsity level needed to use sparse feature representation Default value:'0.7' (short form sp)
`featureFirstUsePenalty`	double: <float>. The feature first use penalty coefficient Default value:'0' (short form ffup)
`featureReusePenalty`	double: <float>. The feature re-use penalty (regularization) coefficient Default value:'0' (short form frup)
`gainConfidenceLevel`	double: <float>. Tree fitting gain confidence requirement (should be in the range [0,1)). Default value:'0' (short form gainconf)

`softmaxTemperature`	double: <float>. The temperature of the randomized softmax distribution for choosing the feature Default value:'0' (short form smtemp)
`executionTimes`	logical: [+|-]. Print execution time breakdown to stdout Default value:'-' (short form et)
`numLeaves`	integer: <int>. The max number of leaves in each regression tree Default value:'20' (short form nl)
`minDocumentsInLeafs`	integer: <int>. The minimal number of documents allowed in a leaf of a regression tree, out of the subsampled data Default value:'10' (short form mil)
`numTrees`	integer: <int>. Number of weak hypotheses in the ensemble Default value:'100' (short form iter)
`smoothing`	double: <float>. Smoothing paramter for tree regularization Default value: '0.0' (short form s)
`testFrequency`	integer: <int>. Calculate metric values for train/valid/test every k rounds Default value: '2147483647' (short form tf)
`baggingSize`	integer: <int>. Number of trees in each bag (0 for disabling bagging) Default value: '0' (short form bag)
`...`	: . hidden arguments

Value

a character string defining: FastForestRegression (RegressorTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (quantileSampleCount = 100,
    featureFraction = 0.7, baggingTrainFraction = 0.7, splitFraction = 0.7,
    numThreads = 8, rngSeed = 123, entropyCoefficient = 0,
    histogramPoolSize = -1, diskTranspose = FALSE, maxBins = 255,
    sparsifyThreshold = 0.7, featureFirstUsePenalty = 0, featureReusePenalty = 0,
    gainConfidenceLevel = 0, softmaxTemperature = 0, executionTimes = FALSE,
    numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100, smoothing = 0.0,
    testFrequency = 2147483647, baggingSize = 1, ...)
{
    params <- character()
    params <- mluPasteArg(quantileSampleCount, "integer", params)
    params <- mluPasteArg(featureFraction, "double", params)
    params <- mluPasteArg(baggingTrainFraction, "double", params)
    params <- mluPasteArg(splitFraction, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(rngSeed, "integer", params)
    params <- mluPasteArg(entropyCoefficient, "double", params)
    params <- mluPasteArg(histogramPoolSize, "integer", params)
    params <- mluPasteArg(diskTranspose, "logical", params)
    params <- mluPasteArg(maxBins, "integer", params)
    params <- mluPasteArg(sparsifyThreshold, "double", params)
    params <- mluPasteArg(featureFirstUsePenalty, "double", params)
    params <- mluPasteArg(featureReusePenalty, "double", params)
    params <- mluPasteArg(gainConfidenceLevel, "double", params)
    params <- mluPasteArg(softmaxTemperature, "double", params)
    params <- mluPasteArg(executionTimes, "logical", params)
    params <- mluPasteArg(numLeaves, "integer", params)
    params <- mluPasteArg(minDocumentsInLeafs, "integer", params)
    params <- mluPasteArg(numTrees, "integer", params)
    params <- mluPasteArg(smoothing, 'double', params)
    params <- mluPasteArg(testFrequency, 'integer', params)
    params <- mluPasteArg(baggingSize, 'integer', params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("FastForestRegression{%s}", params)
    return(structure(params, class = c("FastForestRegression",
        "RegressorTrainer", "maml", "character")))
  }
```

---

tlcFastTreeBinaryClassification

BinaryClassifierTrainer, Trainer: 'FastTreeBinaryClassification'

Description

Uses a logit-boost boosted tree learner to perform binary classification.

Usage

```
tlcFastTreeBinaryClassification(unbalancedSets = FALSE, featureFraction = 1,
  baggingSize = 0, baggingTrainFraction = 0.7,
  bestStepRankingRegressionTrees = FALSE, useLineSearch = FALSE,
  numPostBracketSteps = 0, minStepSize = 0,
  optimizationAlgorithm = "GradientDescent", earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, enablePruning = FALSE,
  useTolerantPruning = FALSE, pruningThreshold = 0.004,
  pruningWindowSize = 5, testFrequency = 2147483647, learningRates = 0.2,
  shrinkage = 1, dropoutRate = 0, splitFraction = 1,
  getDerivativesSampleRate = 1, writeLastEnsemble = FALSE, smoothing = 0,
  maxTreeOutput = 100, numThreads = 8, rngSeed = 123,
  fileSplitSeed = 123, entropyCoefficient = 0, histogramPoolSize = -1,
  diskTranspose = FALSE, maxBins = 255, sparsifyThreshold = 0.7,
  featureFirstUsePenalty = 0, featureReusePenalty = 0,
  gainConfidenceLevel = 0, softmaxTemperature = 0, executionTimes = FALSE,
  numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100)
```

Arguments

`unbalancedSets`	logical: [+|-]. Should we use derivatives optimized for unbalanced sets Default value:'-' (short form us)
`featureFraction`	double: <float>. The fraction of features (chosen randomly) to use on each iteration Default value:'1' (short form ff)
`baggingSize`	integer: <int>. Number of trees in each bag (0 for disabling bagging) Default value:'0' (short form bag)
`baggingTrainFraction`	double: <float>. Percentage of training queries used in each bag Default value:'0.7' (short form bagfrac)
`bestStepRankingRegressionTrees`	logical: [+|-]. Use best regression step trees? Default value:'-' (short form bsr)
`useLineSearch`	logical: [+|-]. Should we use line search for a step size Default value:'-' (short form ls)
`numPostBracketSteps`	integer: <int>. Number of post-bracket line search steps Default value:'0' (short form lssteps)
`minStepSize`	double: <float>. Minimum line search step size Default value:'0' (short form minstep)
`optimizationAlgorithm`	character: [GradientDescent|AcceleratedGradientDescent|ConjugateGradientDescent]. Optimization algorithm to be used (GradientDescent, AcceleratedGradientDescent) Default value:'GradientDescent' (short form oa)
`earlyStoppingRule`	list: <name><options>. Early stopping rule. (Validation set (/valid) is required.) (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics. (For regression, 1: L1, 2:L2; for ranking, 1:NDCG@1, 3:NDCG@3) Default value:'0' (short form esmt)
`enablePruning`	logical: [+|-]. Enable post-training pruning to avoid overfitting. (a validation set is required) Default value:'-' (short form pruning)
`useTolerantPruning`	logical: [+|-]. Use window and tolerance for pruning Default value:'-' (short form prtol)
`pruningThreshold`	double: <double>. The tolerance threshold for pruning Default value:'0.004' (short form prth)
`pruningWindowSize`	integer: <int>. The moving window size for pruning Default value:'5' (short form prws)
`testFrequency`	integer: <int>. Calculate NDCG values for train/valid/test every k rounds Default value:'2147483647' (short form tf)
`learningRates`	double: <float>. The learning rate Default value:'0.2' (short form lr)
`shrinkage`	double: <float>. Shrinkage Default value:'1' (short form shrk)

`dropoutRate`	double: <float>. Dropout rate for tree regularization Default value:'0' (short form tdrop)
`splitFraction`	double: <float>. The fraction of features (chosen randomly) to use on each split Default value:'1' (short form sf)
`getDerivativesSampleRate`	integer: <int>. same each query 1 in k times in the GetDerivatives function Default value:'1' (short form sr)
`writeLastEnsemble`	logical: [+|-]. Write the last ensemble instead of the one determined by early stopping Default value:'-' (short form hl)
`smoothing`	double: <float>. Smoothing paramter for tree regularization Default value:'0' (short form s)
`maxTreeOutput`	double: <float>. Upper bound on absolute value of single tree output Default value:'100' (short form mo)
`numThreads`	integer: <int>. The number of threads to use Default value:'8' (short form t)
`rngSeed`	integer: <int>. The seed of the random number generator Default value:'123' (short form r1)
`fileSplitSeed`	integer: <int>. The seed of the file splitter Default value:'123' (short form r2)
`entropyCoefficient`	double: <float>. The entropy (regularization) coefficient between 0 and 1 Default value:'0' (short form e)
`histogramPoolSize`	integer: <int>. The number of histograms in the pool (between 2 and numLeaves) Default value:'-1' (short form ps)
`diskTranspose`	logical: [+|-]. Whether to utilize the disk when performing the transpose Default value:'-' (short form dt)
`maxBins`	integer: <int>. Maximum number of distinct values (bins) per feature Default value:'255' (short form mb)
`sparsifyThreshold`	double: <float>. Sparsity level needed to use sparse feature representation Default value:'0.7' (short form sp)
`featureFirstUsePenalty`	double: <float>. The feature first use penalty coefficient Default value:'0' (short form ffup)
`featureReusePenalty`	double: <float>. The feature re-use penalty (regularization) coefficient Default value:'0' (short form frup)
`gainConfidenceLevel`	double: <float>. Tree fitting gain confidence requirement (should be in the range [0,1)). Default value:'0' (short form gainconf)
`softmaxTemperature`	double: <float>. The temperature of the randomized softmax distribution for choosing the feature Default value:'0' (short form smtemp)
`executionTimes`	logical: [+|-]. Print execution time breakdown to stdout Default value:'-' (short form et)
`numLeaves`	integer: <int>. The max number of leaves in each regression tree Default value:'20' (short form nl)

`minDocumentsInLeafs`	integer: <int>. The minimal number of documents allowed in a leaf of a regression tree, out of the subsampled data Default value:'10' (short form mil)
`numTrees`	integer: <int>. Number of weak hypotheses in the ensemble Default value:'100' (short form iter)
`...`	: . hidden arguments

Value

a character string defining: FastTreeBinaryClassification (BinaryClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (unbalancedSets = FALSE, featureFraction = 1, baggingSize = 0,
    baggingTrainFraction = 0.7, bestStepRankingRegressionTrees = FALSE,
    useLineSearch = FALSE, numPostBracketSteps = 0, minStepSize = 0,
    optimizationAlgorithm = "GradientDescent", earlyStoppingRule = list(),
    earlyStoppingMetrics = 0, enablePruning = FALSE, useTolerantPruning = FALSE,
    pruningThreshold = 0.004, pruningWindowSize = 5, testFrequency = 2147483647,
    learningRates = 0.2, shrinkage = 1, dropoutRate = 0, splitFraction = 1,
    getDerivativesSampleRate = 1, writeLastEnsemble = FALSE,
    smoothing = 0, maxTreeOutput = 100,
    numThreads = 8, rngSeed = 123,
    fileSplitSeed = 123, entropyCoefficient = 0, histogramPoolSize = -1,
    diskTranspose = FALSE, maxBins = 255, sparsifyThreshold = 0.7,
    featureFirstUsePenalty = 0, featureReusePenalty = 0, gainConfidenceLevel = 0,
    softmaxTemperature = 0, executionTimes = FALSE,
    numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100,
    ...)
{
    params <- character()
    params <- mluPasteArg(unbalancedSets, "logical", params)
    params <- mluPasteArg(featureFraction, "double", params)
    params <- mluPasteArg(baggingSize, "integer", params)
    params <- mluPasteArg(baggingTrainFraction, "double", params)
    params <- mluPasteArg(bestStepRankingRegressionTrees, "logical", params)
    params <- mluPasteArg(useLineSearch, "logical", params)
    params <- mluPasteArg(numPostBracketSteps, "integer", params)
    params <- mluPasteArg(minStepSize, "double", params)
    params <- mluPasteArg(optimizationAlgorithm, "character", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(enablePruning, "logical", params)
    params <- mluPasteArg(useTolerantPruning, "logical", params)
    params <- mluPasteArg(pruningThreshold, "double", params)
    params <- mluPasteArg(pruningWindowSize, "integer", params)
    params <- mluPasteArg(testFrequency, "integer", params)
    params <- mluPasteArg(learningRates, "double", params)
    params <- mluPasteArg(shrinkage, "double", params)
    params <- mluPasteArg(dropoutRate, "double", params)
    params <- mluPasteArg(splitFraction, "double", params)
    params <- mluPasteArg(getDerivativesSampleRate, "integer", params)
    params <- mluPasteArg(writeLastEnsemble, "logical", params)
    params <- mluPasteArg(smoothing, "double", params)
    params <- mluPasteArg(maxTreeOutput, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(rngSeed, "integer", params)
    params <- mluPasteArg(fileSplitSeed, "integer", params)
    params <- mluPasteArg(entropyCoefficient, "double", params)
    params <- mluPasteArg(histogramPoolSize, "integer", params)
    params <- mluPasteArg(diskTranspose, "logical", params)
    params <- mluPasteArg(maxBins, "integer", params)
    params <- mluPasteArg(sparsifyThreshold, "double", params)
    params <- mluPasteArg(featureFirstUsePenalty, "double", params)
    params <- mluPasteArg(featureReusePenalty, "double", params)
    params <- mluPasteArg(gainConfidenceLevel, "double", params)
    params <- mluPasteArg(softmaxTemperature, "double", params)
    params <- mluPasteArg(executionTimes, "logical", params)
    params <- mluPasteArg(numLeaves, "integer", params)
    params <- mluPasteArg(minDocumentsInLeafs, "integer", params)
    params <- mluPasteArg(numTrees, "integer", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("FastTreeBinaryClassification{%s}", params)
    return(structure(params, class = c("FastTreeBinaryClassification",
        "BinaryClassifierTrainer", "maml", "character")))
}
```

`tlcFastTreeRanking` *RankerTrainer, Trainer: 'FastTreeRanking'*

Description

Trains gradient boosted decision trees to the LambdaRank quasi-gradient.

Usage

```
tlcFastTreeRanking(customGains = "0,3,7,15,31", trainDCG = FALSE,
  featureFraction = 1, baggingSize = 0, baggingTrainFraction = 0.7,
  bestStepRankingRegressionTrees = FALSE, useLineSearch = FALSE,
  numPostBracketSteps = 0, minStepSize = 0,
  optimizationAlgorithm = "GradientDescent", earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, enablePruning = FALSE,
  useTolerantPruning = FALSE, pruningThreshold = 0.004,
  pruningWindowSize = 5, testFrequency = 2147483647, learningRates = 0.2,
  shrinkage = 1, dropoutRate = 0, splitFraction = 1,
  getDerivativesSampleRate = 1, writeLastEnsemble = FALSE, smoothing = 0,
  maxTreeOutput = 100, numThreads = 8, rngSeed = 123,
  entropyCoefficient = 0, histogramPoolSize = -1, diskTranspose = FALSE,
  maxBins = 255, sparsifyThreshold = 0.7, featureFirstUsePenalty = 0,
  featureReusePenalty = 0, gainConfidenceLevel = 0,
  softmaxTemperature = 0, executionTimes = FALSE, numLeaves = 20,
  minDocumentsInLeafs = 10, numTrees = 100, ...)
```

Arguments

`customGains`	character: <string>. Comma seperated list of gains associated to each relevance label. Default value:'0,3,7,15,31' (short form gains)
`trainDCG`	logical: [+|-]. Train DCG instead of NDCG Default value:'-' (short form dcg)
`featureFraction`	double: <float>. The fraction of features (chosen randomly) to use on each iteration Default value:'1' (short form ff)
`baggingSize`	integer: <int>. Number of trees in each bag (0 for disabling bagging) Default value:'0' (short form bag)
`baggingTrainFraction`	double: <float>. Percentage of training queries used in each bag Default value:'0.7' (short form bagfrac)
`bestStepRankingRegressionTrees`	logical: [+|-]. Use best regression step trees? Default value:'-' (short form bsr)
`useLineSearch`	logical: [+|-]. Should we use line search for a step size Default value:'-' (short form ls)
`numPostBracketSteps`	integer: <int>. Number of post-bracket line search steps Default value:'0' (short form lssteps)
`minStepSize`	double: <float>. Minimum line search step size Default value:'0' (short form minstep)
`optimizationAlgorithm`	character: [GradientDescent|AcceleratedGradientDescent|ConjugateGradientDescent]. Optimization algorithm to be used (GradientDescent, AcceleratedGradientDescent) Default value:'GradientDescent' (short form oa)
`earlyStoppingRule`	list: <name><options>. Early stopping rule. (Validation set (/valid) is required.) (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics. (For regression, 1: L1, 2:L2; for ranking, 1:NDCG@1, 3:NDCG@3) Default value:'0' (short form esmt)
`enablePruning`	logical: [+|-]. Enable post-training pruning to avoid overfitting. (a validation set is required) Default value:'-' (short form pruning)
`useTolerantPruning`	logical: [+|-]. Use window and tolerance for pruning Default value:'-' (short form prtol)

`pruningThreshold`	double: <double>. The tolerance threshold for pruning Default value:'0.004' (short form prth)
`pruningWindowSize`	integer: <int>. The moving window size for pruning Default value:'5' (short form prws)
`testFrequency`	integer: <int>. Calculate NDCG values for train/valid/test every k rounds Default value:'2147483647' (short form tf)
`learningRates`	double: <float>. The learning rate Default value:'0.2' (short form lr)
`shrinkage`	double: <float>. Shrinkage Default value:'1' (short form shrk)
`dropoutRate`	double: <float>. Dropout rate for tree regularization Default value:'0' (short form tdrop)
`splitFraction`	double: <float>. The fraction of features (chosen randomly) to use on each split Default value:'1' (short form sf)
`getDerivativesSampleRate`	integer: <int>. same each query 1 in k times in the GetDerivatives function Default value:'1' (short form sr)
`writeLastEnsemble`	logical: [+|-]. Write the last ensemble instead of the one determined by early stopping Default value:'-' (short form hl)
`smoothing`	double: <float>. Smoothing paramter for tree regularization Default value:'0' (short form s)
`maxTreeOutput`	double: <float>. Upper bound on absolute value of single tree output Default value:'100' (short form mo)
`numThreads`	integer: <int>. The number of threads to use Default value:'8' (short form t)
`rngSeed`	integer: <int>. The seed of the random number generator Default value:'123' (short form r1)
`entropyCoefficient`	double: <float>. The entropy (regularization) coefficient between 0 and 1 Default value:'0' (short form e)
`histogramPoolSize`	integer: <int>. The number of histograms in the pool (between 2 and numLeaves) Default value:'-1' (short form ps)
`diskTranspose`	logical: [+|-]. Whether to utilize the disk when performing the transpose Default value:'-' (short form dt)
`maxBins`	integer: <int>. Maximum number of distinct values (bins) per feature Default value:'255' (short form mb)
`sparsifyThreshold`	double: <float>. Sparsity level needed to use sparse feature representation Default value:'0.7' (short form sp)
`featureFirstUsePenalty`	double: <float>. The feature first use penalty coefficient Default value:'0' (short form ffup)
`featureReusePenalty`	double: <float>. The feature re-use penalty (regularization) coefficient Default value:'0' (short form frup)
`gainConfidenceLevel`	double: <float>. Tree fitting gain confidence requirement (should be in the range [0,1)). Default value:'0' (short form gainconf)

`softmaxTemperature`	double: <float>. The temperature of the randomized softmax distribution for choosing the feature Default value:'0' (short form smtemp)
`executionTimes`	logical: [+|-]. Print execution time breakdown to stdout Default value:'-' (short form et)
`numLeaves`	integer: <int>. The max number of leaves in each regression tree Default value:'20' (short form nl)
`minDocumentsInLeafs`	integer: <int>. The minimal number of documents allowed in a leaf of a regression tree, out of the subsampled data Default value:'10' (short form mil)
`numTrees`	integer: <int>. Number of weak hypotheses in the ensemble Default value:'100' (short form iter)
`...`	: . hidden arguments

Value

a character string defining: FastTreeRanking (RankerTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (customGains = "0,3,7,15,31", trainDCG = FALSE, featureFraction = 1,
    baggingSize = 0, baggingTrainFraction = 0.7, bestStepRankingRegressionTrees = FALSE,
    useLineSearch = FALSE, numPostBracketSteps = 0, minStepSize = 0,
    optimizationAlgorithm = "GradientDescent", earlyStoppingRule = list(),
    earlyStoppingMetrics = 0, enablePruning = FALSE, useTolerantPruning = FALSE,
    pruningThreshold = 0.004, pruningWindowSize = 5, testFrequency = 2147483647,
    learningRates = 0.2, shrinkage = 1, dropoutRate = 0, splitFraction = 1,
    getDerivativesSampleRate = 1, writeLastEnsemble = FALSE,
    smoothing = 0, maxTreeOutput = 100,
    numThreads = 8, rngSeed = 123,
    entropyCoefficient = 0, histogramPoolSize = -1,
    diskTranspose = FALSE, maxBins = 255, sparsifyThreshold = 0.7,
    featureFirstUsePenalty = 0, featureReusePenalty = 0, gainConfidenceLevel = 0,
    softmaxTemperature = 0, executionTimes = FALSE,
    numLeaves = 20, minDocumentsInLeafs = 10, numTrees = 100,
    ...)
{
    params <- character()
    params <- mluPasteArg(customGains, "character", params)
    params <- mluPasteArg(trainDCG, "logical", params)
    params <- mluPasteArg(featureFraction, "double", params)
    params <- mluPasteArg(baggingSize, "integer", params)
    params <- mluPasteArg(baggingTrainFraction, "double", params)
    params <- mluPasteArg(bestStepRankingRegressionTrees, "logical", params)
    params <- mluPasteArg(useLineSearch, "logical", params)
    params <- mluPasteArg(numPostBracketSteps, "integer", params)
    params <- mluPasteArg(minStepSize, "double", params)
    params <- mluPasteArg(optimizationAlgorithm, "character", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(enablePruning, "logical", params)
    params <- mluPasteArg(useTolerantPruning, "logical", params)
    params <- mluPasteArg(pruningThreshold, "double", params)
    params <- mluPasteArg(pruningWindowSize, "integer", params)
    params <- mluPasteArg(testFrequency, "integer", params)
    params <- mluPasteArg(learningRates, "double", params)
    params <- mluPasteArg(shrinkage, "double", params)
    params <- mluPasteArg(dropoutRate, "double", params)
    params <- mluPasteArg(splitFraction, "double", params)
    params <- mluPasteArg(getDerivativesSampleRate, "integer", params)
    params <- mluPasteArg(writeLastEnsemble, "logical", params)
    params <- mluPasteArg(smoothing, "double", params)
    params <- mluPasteArg(maxTreeOutput, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(rngSeed, "integer", params)
    params <- mluPasteArg(entropyCoefficient, "double", params)
    params <- mluPasteArg(histogramPoolSize, "integer", params)
    params <- mluPasteArg(diskTranspose, "logical", params)
    params <- mluPasteArg(maxBins, "integer", params)
    params <- mluPasteArg(sparsifyThreshold, "double", params)
    params <- mluPasteArg(featureFirstUsePenalty, "double", params)
    params <- mluPasteArg(featureReusePenalty, "double", params)
    params <- mluPasteArg(gainConfidenceLevel, "double", params)
    params <- mluPasteArg(softmaxTemperature, "double", params)
    params <- mluPasteArg(executionTimes, "logical", params)
    params <- mluPasteArg(numLeaves, "integer", params)
    params <- mluPasteArg(minDocumentsInLeafs, "integer", params)
    params <- mluPasteArg(numTrees, "integer", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("FastTreeRanking{%s}", params)
    return(structure(params, class = c("FastTreeRanking",
"RankerTrainer", "maml", "character")))
  }
```

tlcFastTreeRegression *RegressorTrainer, Trainer: 'FastTreeRegression'*

Description

Trains gradient boosted decision trees to fit target values using least-squares.

Usage

```
tlcFastTreeRegression(featureFraction = 1, baggingSize = 0,
  baggingTrainFraction = 0.7, bestStepRankingRegressionTrees = FALSE,
  useLineSearch = FALSE, numPostBracketSteps = 0, minStepSize = 0,
  optimizationAlgorithm = "GradientDescent", earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, enablePruning = FALSE,
  useTolerantPruning = FALSE, pruningThreshold = 0.004,
  pruningWindowSize = 5, testFrequency = 2147483647, learningRates = 0.2,
  shrinkage = 1, dropoutRate = 0, splitFraction = 1,
  getDerivativesSampleRate = 1, writeLastEnsemble = FALSE, smoothing = 0,
  maxTreeOutput = 100, numThreads = 8, rngSeed = 123,
  entropyCoefficient = 0, histogramPoolSize = -1, diskTranspose = FALSE,
  maxBins = 255, sparsifyThreshold = 0.7, featureFirstUsePenalty = 0,
  featureReusePenalty = 0, gainConfidenceLevel = 0,
  softmaxTemperature = 0, executionTimes = FALSE, numLeaves = 20,
  minDocumentsInLeafs = 10, numTrees = 100, ...)
```

Arguments

`featureFraction`	double: <float>. The fraction of features (chosen randomly) to use on each iteration Default value:'1' (short form ff)
`baggingSize`	integer: <int>. Number of trees in each bag (0 for disabling bagging) Default value:'0' (short form bag)
`baggingTrainFraction`	double: <float>. Percentage of training queries used in each bag Default value:'0.7' (short form bagfrac)
`bestStepRankingRegressionTrees`	logical: [+|-]. Use best regression step trees? Default value:'-' (short form bsr)
`useLineSearch`	logical: [+|-]. Should we use line search for a step size Default value:'-' (short form ls)
`numPostBracketSteps`	integer: <int>. Number of post-bracket line search steps Default value:'0' (short form lssteps)
`minStepSize`	double: <float>. Minimum line search step size Default value:'0' (short form minstep)
`optimizationAlgorithm`	character: [GradientDescent|AcceleratedGradientDescent|ConjugateGradientDescent]. Optimization algorithm to be used (GradientDescent, AcceleratedGradientDescent) Default value:'GradientDescent' (short form oa)
`earlyStoppingRule`	list: <name><options>. Early stopping rule. (Validation set (/valid) is required.) (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics. (For regression, 1: L1, 2:L2; for ranking, 1:NDCG@1, 3:NDCG@3) Default value:'0' (short form esmt)
`enablePruning`	logical: [+|-]. Enable post-training pruning to avoid overfitting. (a validation set is required) Default value:'-' (short form pruning)
`useTolerantPruning`	logical: [+|-]. Use window and tolerance for pruning Default value:'-' (short form prtol)

`pruningThreshold`	double: <double>. The tolerance threshold for pruning Default value:'0.004' (short form prth)
`pruningWindowSize`	integer: <int>. The moving window size for pruning Default value:'5' (short form prws)
`testFrequency`	integer: <int>. Calculate NDCG values for train/valid/test every k rounds Default value:'2147483647' (short form tf)
`learningRates`	double: <float>. The learning rate Default value:'0.2' (short form lr)
`shrinkage`	double: <float>. Shrinkage Default value:'1' (short form shrk)
`dropoutRate`	double: <float>. Dropout rate for tree regularization Default value:'0' (short form tdrop)
`splitFraction`	double: <float>. The fraction of features (chosen randomly) to use on each split Default value:'1' (short form sf)
`getDerivativesSampleRate`	integer: <int>. same each query 1 in k times in the GetDerivatives function Default value:'1' (short form sr)
`writeLastEnsemble`	logical: [+|-]. Write the last ensemble instead of the one determined by early stopping Default value:'-' (short form hl)
`smoothing`	double: <float>. Smoothing paramter for tree regularization Default value:'0' (short form s)
`maxTreeOutput`	double: <float>. Upper bound on absolute value of single tree output Default value:'100' (short form mo)
`numThreads`	integer: <int>. The number of threads to use Default value:'8' (short form t)
`rngSeed`	integer: <int>. The seed of the random number generator Default value:'123' (short form r1)
`entropyCoefficient`	double: <float>. The entropy (regularization) coefficient between 0 and 1 Default value:'0' (short form e)
`histogramPoolSize`	integer: <int>. The number of histograms in the pool (between 2 and numLeaves) Default value:'-1' (short form ps)
`diskTranspose`	logical: [+|-]. Whether to utilize the disk when performing the transpose Default value:'-' (short form dt)
`maxBins`	integer: <int>. Maximum number of distinct values (bins) per feature Default value:'255' (short form mb)
`sparsifyThreshold`	double: <float>. Sparsity level needed to use sparse feature representation Default value:'0.7' (short form sp)
`featureFirstUsePenalty`	double: <float>. The feature first use penalty coefficient Default value:'0' (short form ffup)
`featureReusePenalty`	double: <float>. The feature re-use penalty (regularization) coefficient Default value:'0' (short form frup)
`gainConfidenceLevel`	double: <float>. Tree fitting gain confidence requirement (should be in the range [0,1)). Default value:'0' (short form gainconf)

`softmaxTemperature`	double: <float>. The temperature of the randomized softmax distribution for choosing the feature Default value:'0' (short form smtemp)
`executionTimes`	logical: [+|-]. Print execution time breakdown to stdout Default value:'-' (short form et)
`numLeaves`	integer: <int>. The max number of leaves in each regression tree Default value:'20' (short form nl)
`minDocumentsInLeafs`	integer: <int>. The minimal number of documents allowed in a leaf of a regression tree, out of the subsampled data Default value:'10' (short form mil)
`numTrees`	integer: <int>. Number of weak hypotheses in the ensemble Default value:'100' (short form iter)
`...`	: . hidden arguments

Value

a character string defining: FastTreeRegression (RegressorTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (featureFraction = 1, baggingSize = 0, baggingTrainFraction = 0.7,
    bestStepRankingRegressionTrees = FALSE, useLineSearch = FALSE,
    numPostBracketSteps = 0, minStepSize = 0, optimizationAlgorithm = "GradientDescent",
    earlyStoppingRule = list(), earlyStoppingMetrics = 0, enablePruning = FALSE,
    useTolerantPruning = FALSE, pruningThreshold = 0.004, pruningWindowSize = 5,
    testFrequency = 2147483647, learningRates = 0.2, shrinkage = 1,
    dropoutRate = 0, splitFraction = 1, getDerivativesSampleRate = 1,
    writeLastEnsemble = FALSE, smoothing = 0, maxTreeOutput = 100, numThreads = 8,
    rngSeed = 123, entropyCoefficient = 0,
    histogramPoolSize = -1, diskTranspose = FALSE, maxBins = 255,
    sparsifyThreshold = 0.7, featureFirstUsePenalty = 0, featureReusePenalty = 0,
    gainConfidenceLevel = 0, softmaxTemperature = 0, executionTimes = FALSE,
    numLeaves = 20, minDocumentsInLeafs = 10,
    numTrees = 100, ...)
{
    params <- character()
    params <- mluPasteArg(featureFraction, "double", params)
    params <- mluPasteArg(baggingSize, "integer", params)
    params <- mluPasteArg(baggingTrainFraction, "double", params)
    params <- mluPasteArg(bestStepRankingRegressionTrees, "logical", params)
    params <- mluPasteArg(useLineSearch, "logical", params)
    params <- mluPasteArg(numPostBracketSteps, "integer", params)
    params <- mluPasteArg(minStepSize, "double", params)
    params <- mluPasteArg(optimizationAlgorithm, "character", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(enablePruning, "logical", params)
    params <- mluPasteArg(useTolerantPruning, "logical", params)
    params <- mluPasteArg(pruningThreshold, "double", params)
    params <- mluPasteArg(pruningWindowSize, "integer", params)
    params <- mluPasteArg(testFrequency, "integer", params)
    params <- mluPasteArg(learningRates, "double", params)
    params <- mluPasteArg(shrinkage, "double", params)
    params <- mluPasteArg(dropoutRate, "double", params)
    params <- mluPasteArg(splitFraction, "double", params)
    params <- mluPasteArg(getDerivativesSampleRate, "integer", params)
    params <- mluPasteArg(writeLastEnsemble, "logical", params)
    params <- mluPasteArg(smoothing, "double", params)
    params <- mluPasteArg(maxTreeOutput, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(rngSeed, "integer", params)
    params <- mluPasteArg(entropyCoefficient, "double", params)
    params <- mluPasteArg(histogramPoolSize, "integer", params)
    params <- mluPasteArg(diskTranspose, "logical", params)
    params <- mluPasteArg(maxBins, "integer", params)
    params <- mluPasteArg(sparsifyThreshold, "double", params)
    params <- mluPasteArg(featureFirstUsePenalty, "double", params)
    params <- mluPasteArg(featureReusePenalty, "double", params)
    params <- mluPasteArg(gainConfidenceLevel, "double", params)
    params <- mluPasteArg(softmaxTemperature, "double", params)
    params <- mluPasteArg(executionTimes, "logical", params)
    params <- mluPasteArg(numLeaves, "integer", params)
    params <- mluPasteArg(minDocumentsInLeafs, "integer", params)
    params <- mluPasteArg(numTrees, "integer", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("FastTreeRegression{%s}", params)
    return(structure(params, class = c("FastTreeRegression",
        "RegressorTrainer", "maml", "character")))
}
```

`tlcLogisticRegression` *BinaryClassifierTrainer, Trainer: 'LogisticRegression'*

Description

Logistic Regression is a method in statistics used to predict the probability of occurrence of an event and can be used as a classification algorithm. The algorithm predicts the probability of occurrence of an event by fitting data to a logistical function.

Usage

```
tlcLogisticRegression(l2Weight = 1, l1Weight = 1, optTol = 1e-07,
  memorySize = 20, maxIterations = 2147483647, showTrainingStats = FALSE,
  sgdInitializationTolerance = 0, quiet = FALSE, initWtsDiameter = 0,
  numThreads = NULL, denseOptimizer = FALSE)
```

Arguments

`l2Weight`	double: <float>. L2 regularization weight Default value:'1' (short form l2)
`l1Weight`	double: <float>. L1 regularization weight Default value:'1' (short form l1)
`optTol`	double: <float>. Tolerance parameter for optimization convergence. Lower = slower, more accurate Default value:'1E-07' (short form ot)
`memorySize`	integer: <int>. Memory size for L-BFGS. Lower=faster, less accurate Default value:'20' (short form m)
`maxIterations`	integer: <int>. Maximum iterations. Default value:'2147483647' (short form maxiter)
`showTrainingStats`	logical: [+|-]. Include training statistics in model Default value:'-'
`sgdInitializationTolerance`	double: <float>. Run SGD to initialize LR weights, converging to this tolerance Default value:'0' (short form sgd)
`quiet`	logical: [+|-]. If set to true, produce no output during training. Default value:'-' (short form q)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0' (short form initwts)
`numThreads`	integer: <int>. Number of threads (short form nt)
`denseOptimizer`	logical: [+|-]. Force densification of the internal optimization vectors Default value:'-' (short form do)
`...`	: . hidden arguments

Value

a character string defining: LogisticRegression (BinaryClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (l2Weight = 1, l1Weight = 1, optTol = 1e-07, memorySize = 20,
   maxIterations = 2147483647, showTrainingStats = FALSE, sgdInitializationTolerance = 0,
    quiet = FALSE, initWtsDiameter = 0, numThreads = NULL, denseOptimizer = FALSE,
    ...)
{
    params <- character()
    params <- mluPasteArg(l2Weight, "double", params)
    params <- mluPasteArg(l1Weight, "double", params)
    params <- mluPasteArg(optTol, "double", params)
    params <- mluPasteArg(memorySize, "integer", params)
    params <- mluPasteArg(maxIterations, "integer", params)
    params <- mluPasteArg(showTrainingStats, "logical", params)
    params <- mluPasteArg(sgdInitializationTolerance, "double", params)
    params <- mluPasteArg(quiet, "logical", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(denseOptimizer, "logical", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("LogisticRegression{%s}", params)
    return(structure(params, class = c("LogisticRegression",
        "BinaryClassifierTrainer", "maml", "character")))
  }
```

---

tlcMultiClassLogisticRegression

MultiClassClassifierTrainer, Trainer: 'MultiClassLogisticRegression'

Description

Not Available!

Usage

```
tlcMultiClassLogisticRegression(l2Weight = 1, l1Weight = 1,
  optTol = 1e-07, memorySize = 20, maxIterations = 2147483647,
  showTrainingStats = FALSE, sgdInitializationTolerance = 0,
  quiet = FALSE, initWtsDiameter = 0, numThreads = NULL,
  denseOptimizer = FALSE)
```

Arguments

`l2Weight`	double: <float>. L2 regularization weight Default value:'1' (short form l2)
`l1Weight`	double: <float>. L1 regularization weight Default value:'1' (short form l1)
`optTol`	double: <float>. Tolerance parameter for optimization convergence. Lower = slower, more accurate Default value:'1E-07' (short form ot)
`memorySize`	integer: <int>. Memory size for L-BFGS. Lower=faster, less accurate Default value:'20' (short form m)
`maxIterations`	integer: <int>. Maximum iterations. Default value:'2147483647' (short form maxiter)

`showTrainingStats`	logical: [+|-]. Include training statistics in model Default value:'-'
`sgdInitializationTolerance`	double: <float>. Run SGD to initialize LR weights, converging to this tolerance Default value:'0' (short form sgd)
`quiet`	logical: [+|-]. If set to true, produce no output during training. Default value:'-' (short form q)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0' (short form initwts)
`numThreads`	integer: <int>. Number of threads (short form nt)
`denseOptimizer`	logical: [+|-]. Force densification of the internal optimization vectors Default value:'-' (short form do)
`...`	: . hidden arguments

Value

a character string defining: MultiClassLogisticRegression (MultiClassClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (l2Weight = 1, l1Weight = 1, optTol = 1e-07, memorySize = 20,
   maxIterations = 2147483647, showTrainingStats = FALSE, sgdInitializationTolerance = 0,
    quiet = FALSE, initWtsDiameter = 0, numThreads = NULL, denseOptimizer = FALSE,
    ...)
{
    params <- character()
    params <- mluPasteArg(l2Weight, "double", params)
    params <- mluPasteArg(l1Weight, "double", params)
    params <- mluPasteArg(optTol, "double", params)
    params <- mluPasteArg(memorySize, "integer", params)
    params <- mluPasteArg(maxIterations, "integer", params)
    params <- mluPasteArg(showTrainingStats, "logical", params)
    params <- mluPasteArg(sgdInitializationTolerance, "double", params)
    params <- mluPasteArg(quiet, "logical", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(numThreads, "integer", params)
    params <- mluPasteArg(denseOptimizer, "logical", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("MultiClassLogisticRegression{%s}", params)
    return(structure(params, class = c("MultiClassLogisticRegression",
        "MultiClassClassifierTrainer", "maml", "character")))
  }
```

---

tlcMultiClassNeuralNetwork
MultiClassClassifierTrainer, Trainer: 'MultiClassNeuralNetwork'

Description

Multi-class neural network is multi-class classification algorithm which uses neural network with as many outputs as there are classes. TLC supports various types of neural networks, including deep neural networks (DNNs) and convolutional neural networks (CNN) via Net# language.

Usage

```
tlcMultiClassNeuralNetwork(defaultOutputNodes = NULL,
  lossFunction = "CrossEntropy", defaultHiddenNodes = 100,
  netFileName = "", numIterations = 100, displayRefresh = 1,
  optimizationAlgorithm = "sgd", initWtsDiameter = 0.1, maxNorm = 0,
  earlyStoppingRule = list(), earlyStoppingMetrics = 0, pruning = FALSE,
  pruningFactor = 0.01, pruningRounds = 10, pruningRoundIterations = 5,
  acceleration = "avx", preTrainerType = "NoPreTrainer",
  preTrainingEpoch = NULL, miniBatchSize = 1, shuffle = TRUE,
  inputDropoutRate = 0, hiddenDropoutRate = 0, netDefinition = "")
```

Arguments

`defaultOutputNodes`	integer: <int>. Default number of output nodes (short form output)
`lossFunction`	list: <name><options>. Loss function Default value:'CrossEntropy' (short form loss)
`defaultHiddenNodes`	integer: <int>. Default number of hidden nodes Default value:'100' (short form hidden)
`netFileName`	character: <string>. Net file name (short form filename)
`numIterations`	integer: <int>. Number of training iterations Default value:'100' (short form iter)
`displayRefresh`	integer: <int>. Display refresh frequency in number iterations Default value:'1' (short form refresh)
`optimizationAlgorithm`	list: <name><options>. Optimization algorithm (Adadelta or SGD) Default value:'sgd' (short form algo)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0.1' (short form initwts)
`maxNorm`	double: <float>. Constrains the norm of incoming weights of a node Default value:'0'

`earlyStoppingRule`	list: <name><options>. Early stopping rule (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics Default value:'0' (short form esmt)
`pruning`	logical: [+|-]. Enable post-training pruning (Optimal Brain Damage) Default value:'-' (short form prune)
`pruningFactor`	double: <float>. Pruning factor: % of weights removed each pruning iteration Default value:'0.01' (short form prunefact)
`pruningRounds`	integer: <int>. Number of pruning rounds Default value:'10' (short form pruneround)
`pruningRoundIterations`	integer: <int>. Number of pruning round iterations Default value:'5' (short form pruneiter)
`acceleration`	list: <name><options>. Hardware acceleration level Default value:'avx' (short form accel)
`preTrainerType`	character: [NoPreTrainer|Greedy]. Net Pre-Trainer Default value:'NoPreTrainer' (short form pretrain)
`preTrainingEpoch`	integer: <int>. Number of epochs for pre-training. If not set, defaults to numIterations(iter). (short form prepoch)
`miniBatchSize`	integer: <int>. Mini-batch size Default value:'1' (short form mbsize)
`shuffle`	logical: [+|-]. Whether to shuffle for each training iteration Default value:'+' (short form shuf)
`inputDropoutRate`	double: <float>. Input dropout rate Default value:'0' (short form idrop)
`hiddenDropoutRate`	double: <float>. Hidden dropout rate Default value:'0' (short form hdrop)
`netDefinition`	character: <string>. Neural network definition (short form net)
`...`	: . hidden arguments

Value

a character string defining: MultiClassNeuralNetwork (MultiClassClassifierTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
```

```
function (defaultOutputNodes = NULL, lossFunction = "CrossEntropy",
    defaultHiddenNodes = 100, netFileName = "", numIterations = 100,
    displayRefresh = 1, optimizationAlgorithm = "sgd", initWtsDiameter = 0.1,
    maxNorm = 0, earlyStoppingRule = list(), earlyStoppingMetrics = 0,
    pruning = FALSE, pruningFactor = 0.01, pruningRounds = 10,
    pruningRoundIterations = 5, acceleration = "avx", preTrainerType = "NoPreTrainer",
    preTrainingEpoch = NULL, miniBatchSize = 1, shuffle = TRUE,
    inputDropoutRate = 0, hiddenDropoutRate = 0, netDefinition = "", ...)
{
    params <- character()
    params <- mluPasteArg(defaultOutputNodes, "integer", params)
    params <- mluPasteArg(lossFunction, "list", params)
    params <- mluPasteArg(defaultHiddenNodes, "integer", params)
    params <- mluPasteArg(netFileName, "character", params)
    params <- mluPasteArg(numIterations, "integer", params)
    params <- mluPasteArg(displayRefresh, "integer", params)
    params <- mluPasteArg(optimizationAlgorithm, "list", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(maxNorm, "double", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(pruning, "logical", params)
    params <- mluPasteArg(pruningFactor, "double", params)
    params <- mluPasteArg(pruningRounds, "integer", params)
    params <- mluPasteArg(pruningRoundIterations, "integer", params)
    params <- mluPasteArg(acceleration, "list", params)
    params <- mluPasteArg(preTrainerType, "character", params)
    params <- mluPasteArg(preTrainingEpoch, "integer", params)
    params <- mluPasteArg(miniBatchSize, "integer", params)
    params <- mluPasteArg(shuffle, "logical", params)
    params <- mluPasteArg(inputDropoutRate, "double", params)
    params <- mluPasteArg(hiddenDropoutRate, "double", params)
    params <- mluPasteArg(netDefinition, "character", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("MultiClassNeuralNetwork{%s}", params)
    return(structure(params, class = c("MultiClassNeuralNetwork",
        "MultiClassClassifierTrainer", "maml", "character")))
  }
```

---

tlcMultiRegressionNeuralNetwork

MultiOutputRegressorTrainer, Trainer: 'MultiRegressionNeuralNetwork'

Description

Regression neural network is a regression algorithm with multiple outputs.

Usage

```
tlcMultiRegressionNeuralNetwork(lossFunction = "SquaredError",
  defaultHiddenNodes = 100, netFileName = "", numIterations = 100,
  displayRefresh = 1, optimizationAlgorithm = "sgd",
  initWtsDiameter = 0.1, maxNorm = 0, earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, pruning = FALSE, pruningFactor = 0.01,
  pruningRounds = 10, pruningRoundIterations = 5, acceleration = "avx",
  preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
  miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
  hiddenDropoutRate = 0, netDefinition = "", ...)
```

Arguments

`lossFunction`	list: <name><options>. Loss function Default value:'SquaredError' (short form loss)
`defaultHiddenNodes`	integer: <int>. Default number of hidden nodes Default value:'100' (short form hidden)
`netFileName`	character: <string>. Net file name (short form filename)
`numIterations`	integer: <int>. Number of training iterations Default value:'100' (short form iter)
`displayRefresh`	integer: <int>. Display refresh frequency in number iterations Default value:'1' (short form refresh)
`optimizationAlgorithm`	list: <name><options>. Optimization algorithm (Adadelta or SGD) Default value:'sgd' (short form algo)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0.1' (short form initwts)
`maxNorm`	double: <float>. Constrains the norm of incoming weights of a node Default value:'0'
`earlyStoppingRule`	list: <name><options>. Early stopping rule (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics Default value:'0' (short form esmt)
`pruning`	logical: [+|-]. Enable post-training pruning (Optimal Brain Damage) Default value:'-' (short form prune)
`pruningFactor`	double: <float>. Pruning factor: % of weights removed each pruning iteration Default value:'0.01' (short form prunefact)
`pruningRounds`	integer: <int>. Number of pruning rounds Default value:'10' (short form pruneround)
`pruningRoundIterations`	integer: <int>. Number of pruning round iterations Default value:'5' (short form pruneiter)
`acceleration`	list: <name><options>. Hardware acceleration level Default value:'avx' (short form accel)
`preTrainerType`	character: [NoPreTrainer|Greedy]. Net Pre-Trainer Default value:'NoPreTrainer' (short form pretrain)
`preTrainingEpoch`	integer: <int>. Number of epochs for pre-training. If not set, defaults to numIterations(iter). (short form prepoch)
`miniBatchSize`	integer: <int>. Mini-batch size Default value:'1' (short form mbsize)

`shuffle`	logical: [+|-]. Whether to shuffle for each training iteration Default value:'+' (short form shuf)
`inputDropoutRate`	double: <float>. Input dropout rate Default value:'0' (short form idrop)
`hiddenDropoutRate`	double: <float>. Hidden dropout rate Default value:'0' (short form hdrop)
`netDefinition`	character: <string>. Neural network definition (short form net)
`...`	: . hidden arguments

Value

a character string defining: MultiRegressionNeuralNetwork (MultiOutputRegressorTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lossFunction = "SquaredError", defaultHiddenNodes = 100,
    netFileName = "", numIterations = 100, displayRefresh = 1,
    optimizationAlgorithm = "sgd", initWtsDiameter = 0.1, maxNorm = 0,
    earlyStoppingRule = list(), earlyStoppingMetrics = 0, pruning = FALSE,
    pruningFactor = 0.01, pruningRounds = 10, pruningRoundIterations = 5,
    acceleration = "avx", preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
    miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
    hiddenDropoutRate = 0, ...)
{
    params <- character()
    params <- mluPasteArg(lossFunction, "list", params)
    params <- mluPasteArg(defaultHiddenNodes, "integer", params)
    params <- mluPasteArg(netFileName, "character", params)
    params <- mluPasteArg(numIterations, "integer", params)
    params <- mluPasteArg(displayRefresh, "integer", params)
    params <- mluPasteArg(optimizationAlgorithm, "list", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(maxNorm, "double", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(pruning, "logical", params)
    params <- mluPasteArg(pruningFactor, "double", params)
    params <- mluPasteArg(pruningRounds, "integer", params)
    params <- mluPasteArg(pruningRoundIterations, "integer", params)
    params <- mluPasteArg(acceleration, "list", params)
    params <- mluPasteArg(preTrainerType, "character", params)
    params <- mluPasteArg(preTrainingEpoch, "integer", params)
    params <- mluPasteArg(miniBatchSize, "integer", params)
    params <- mluPasteArg(shuffle, "logical", params)
    params <- mluPasteArg(inputDropoutRate, "double", params)
    params <- mluPasteArg(hiddenDropoutRate, "double", params)
    params <- mluPasteArg(netDefinition, 'character', params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("MultiRegressionNeuralNetwork{%s}", params)
    return(structure(params, class = c("MultiRegressionNeuralNetwork",
        "MultiOutputRegressorTrainer", "maml", "character")))
}
```

tlcOneClassSVM *AnomalyDetectorTrainer, Trainer: 'OneClassSVM'*

Description

Not Available!

Usage

```
tlcOneClassSVM(kernel = "RbfKernel", cacheSize = 100, epsilon = 0.001,
  nu = 0.1, shrink = TRUE)
```

Arguments

`kernel`	list: <name><options>. The kernel used for computing inner products Default value:'RbfKernel' (short form ker)
`cacheSize`	double: <double>. Cache size, specified in megabytes Default value:'100' (short form cache)
`epsilon`	double: <double>. Stopping tolerance Default value:'0.001' (short form eps)
`nu`	double: <double>. This parameter determines the trade-off between the fraction of outliers and the number of support vectors Default value:'0.1'
`shrink`	logical: [+|-]. This parameter determines whether or not to use the shrinking heuristic Default value:'+'
`...`	: . hidden arguments

Value

a character string defining: OneClassSVM (AnomalyDetectorTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (kernel = "RbfKernel", cacheSize = 100, epsilon = 0.001,
    nu = 0.1, shrink = TRUE, ...)
{
    params <- character()
    params <- mluPasteArg(kernel, "list", params)
    params <- mluPasteArg(cacheSize, "double", params)
    params <- mluPasteArg(epsilon, "double", params)
    params <- mluPasteArg(nu, "double", params)
    params <- mluPasteArg(shrink, "logical", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("OneClassSVM{%s}", params)
    return(structure(params, class = c("OneClassSvm",
"AnomalyDetectorTrainer", "maml", "character")))
  }
```

`tlcRegressionNeuralNetwork` *RegressorTrainer, Trainer: 'RegressionNeuralNetwork'*

Description

Regression neural network is a regression algorithm which uses neural network with single output. TLC supports various types of neural networks, including deep neural networks (DNNs) and convolutional neural networks (CNN) via Net# language.

Usage

```
tlcRegressionNeuralNetwork(lossFunction = "SquaredError",
  defaultHiddenNodes = 100, netFileName = "", numIterations = 100,
  displayRefresh = 1, optimizationAlgorithm = "sgd",
  initWtsDiameter = 0.1, maxNorm = 0, earlyStoppingRule = list(),
  earlyStoppingMetrics = 0, pruning = FALSE, pruningFactor = 0.01,
  pruningRounds = 10, pruningRoundIterations = 5, acceleration = "avx",
  preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
  miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
  hiddenDropoutRate = 0, netDefinition = "")
```

Arguments

`lossFunction`	list: <name><options>. Loss function Default value:'SquaredError' (short form loss)

`defaultHiddenNodes`	integer: <int>. Default number of hidden nodes Default value:'100' (short form hidden)
`netFileName`	character: <string>. Net file name (short form filename)
`numIterations`	integer: <int>. Number of training iterations Default value:'100' (short form iter)
`displayRefresh`	integer: <int>. Display refresh frequency in number iterations Default value:'1' (short form refresh)
`optimizationAlgorithm`	list: <name><options>. Optimization algorithm (Adadelta or SGD) Default value:'sgd' (short form algo)
`initWtsDiameter`	double: <float>. Init weights diameter Default value:'0.1' (short form initwts)
`maxNorm`	double: <float>. Constrains the norm of incoming weights of a node Default value:'0'
`earlyStoppingRule`	list: <name><options>. Early stopping rule (short form esr)
`earlyStoppingMetrics`	integer: <int>. Early stopping metrics Default value:'0' (short form esmt)
`pruning`	logical: [+|-]. Enable post-training pruning (Optimal Brain Damage) Default value:'-' (short form prune)
`pruningFactor`	double: <float>. Pruning factor: % of weights removed each pruning iteration Default value:'0.01' (short form prunefact)
`pruningRounds`	integer: <int>. Number of pruning rounds Default value:'10' (short form pruneround)
`pruningRoundIterations`	integer: <int>. Number of pruning round iterations Default value:'5' (short form pruneiter)
`acceleration`	list: <name><options>. Hardware acceleration level Default value:'avx' (short form accel)
`preTrainerType`	character: [NoPreTrainer|Greedy]. Net Pre-Trainer Default value:'NoPreTrainer' (short form pretrain)
`preTrainingEpoch`	integer: <int>. Number of epochs for pre-training. If not set, defaults to numIterations(iter). (short form prepoch)
`miniBatchSize`	integer: <int>. Mini-batch size Default value:'1' (short form mbsize)
`shuffle`	logical: [+|-]. Whether to shuffle for each training iteration Default value:'+' (short form shuf)
`inputDropoutRate`	double: <float>. Input dropout rate Default value:'0' (short form idrop)
`hiddenDropoutRate`	double: <float>. Hidden dropout rate Default value:'0' (short form hdrop)
`netDefinition`	character: <string>. Neural network definition (short form net)
`...`	: . hidden arguments

Value

a character string defining: RegressionNeuralNetwork (RegressorTrainer, Trainer).

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (lossFunction = "SquaredError", defaultHiddenNodes = 100,
    netFileName = "", numIterations = 100, displayRefresh = 1,
    optimizationAlgorithm = "sgd", initWtsDiameter = 0.1, maxNorm = 0,
    earlyStoppingRule = list(), earlyStoppingMetrics = 0, pruning = FALSE,
    pruningFactor = 0.01, pruningRounds = 10, pruningRoundIterations = 5,
    acceleration = "avx", preTrainerType = "NoPreTrainer", preTrainingEpoch = NULL,
    miniBatchSize = 1, shuffle = TRUE, inputDropoutRate = 0,
    hiddenDropoutRate = 0, netDefinition = "", ...)
{
    params <- character()
    params <- mluPasteArg(lossFunction, "list", params)
    params <- mluPasteArg(defaultHiddenNodes, "integer", params)
    params <- mluPasteArg(netFileName, "character", params)
    params <- mluPasteArg(numIterations, "integer", params)
    params <- mluPasteArg(displayRefresh, "integer", params)
    params <- mluPasteArg(optimizationAlgorithm, "list", params)
    params <- mluPasteArg(initWtsDiameter, "double", params)
    params <- mluPasteArg(maxNorm, "double", params)
    params <- mluPasteArg(earlyStoppingRule, "list", params)
    params <- mluPasteArg(earlyStoppingMetrics, "integer", params)
    params <- mluPasteArg(pruning, "logical", params)
    params <- mluPasteArg(pruningFactor, "double", params)
    params <- mluPasteArg(pruningRounds, "integer", params)
    params <- mluPasteArg(pruningRoundIterations, "integer", params)
    params <- mluPasteArg(acceleration, "list", params)
    params <- mluPasteArg(preTrainerType, "character", params)
    params <- mluPasteArg(preTrainingEpoch, "integer", params)
    params <- mluPasteArg(miniBatchSize, "integer", params)
    params <- mluPasteArg(shuffle, "logical", params)
    params <- mluPasteArg(inputDropoutRate, "double", params)
    params <- mluPasteArg(hiddenDropoutRate, "double", params)
    params <- mluPasteArg(netDefinition, "character", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    params <- sprintf("RegressionNeuralNetwork{%s}", params)
    return(structure(params, class = c("RegressionNeuralNetwork",
        "RegressorTrainer", "maml", "character")))
  }
```

`tlcScore` *Command: 'Score'*

Description

Scores a data file.

Usage

```
tlcScore(featureColumn = "Features", groupColumn = "GroupId",
  customColumn.with.tag = "", scorer = list(), saver = list(),
  outputDataFile = "", keepHidden = FALSE,
  postTransform.with.tag = list(), outputAllColumns = FALSE,
  outputColumn = "", loader = list(), dataFile = "",
  outputModelFile = "", inputModelFile = "", loadTransforms = FALSE,
  randomSeed = NULL, parallel = NULL, transform.with.tag = list(), ...,
  commandContext = "")
```

Arguments

`featureColumn`	character: <string>. Column to use for features when scorer is not defined Default value:'Features' (short form feat)
`groupColumn`	character: <string>. Group column name Default value:'GroupId' (short form group)
`customColumn.with.tag`	character: <string>. Input columns: Columns with custom kinds declared through key assignments, e.g., col[Kind]=Name to assign column named 'Name' kind 'Kind' (short form col)
`scorer`	list: <name><options>. Scorer to use
`saver`	list: <name><options>. The data saver to use
`outputDataFile`	character: <string>. File to save the data (short form dout)
`keepHidden`	logical: [+|-]. Whether to include hidden columns Default value:'-' (short form keep)
`postTransform.with.tag`	list: <name><options>. Post processing transform (short form pxf)
`outputAllColumns`	logical: [+|-]. Whether to output all columns or just scores (short form all)
`outputColumn`	character: <string>. What columns to output beyond score columns, if outputAllColumns=-. (short form outCol)
`loader`	list: <name><options>. The data loader
`dataFile`	character: <string>. The data file (short form data)
`outputModelFile`	character: <string>. Model file to save (short form out)
`inputModelFile`	character: <string>. Model file to load (short form in)
`loadTransforms`	logical: [+|-]. Load transforms from model file? (short form loadTrans)
`randomSeed`	integer: <int>. Random seed (short form seed)

`parallel`	integer: <int>. Desired degree of parallelism in the data pipeline (short form n)
`transform.with.tag`	list: <name><options>. Transform (short form xf)
`commandContext`	character: [chain|sweep|]. The compute context of the command. Default value:”
`...`	: . hidden arguments

Value

the output of the TLC Command: Score (Command).

Note

args with tag (customColumn.with.tag, postTransform.with.tag, transform.with.tag) can be specified as: structure('...', tag = '...').

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (featureColumn = "Features", groupColumn = "GroupId",
    customColumn.with.tag = "", scorer = list(), saver = list(),
    outputDataFile = "", keepHidden = FALSE, postTransform.with.tag = list(),
    outputAllColumns = FALSE, outputColumn = "", loader = list(),
    dataFile = "", outputModelFile = "", inputModelFile = "",
    loadTransforms = FALSE, randomSeed = NULL, parallel = NULL,
    transform.with.tag = list(), ..., commandContext = "")
{
    params <- character()
    params <- mluPasteArg(featureColumn, "character", params)
    params <- mluPasteArg(groupColumn, "character", params)
    params <- mluPasteArg(customColumn.with.tag, "character", params)
    params <- mluPasteArg(scorer, "list", params)
    params <- mluPasteArg(saver, "list", params)
    params <- mluPasteArg(outputDataFile, "character", params)
    params <- mluPasteArg(keepHidden, "logical", params)
    params <- mluPasteArg(postTransform.with.tag, "list", params)
    params <- mluPasteArg(outputAllColumns, "logical", params)
    params <- mluPasteArg(outputColumn, "character", params)
    params <- mluPasteArg(loader, "list", params)
    params <- mluPasteArg(dataFile, "character", params)
    params <- mluPasteArg(outputModelFile, "character", params)
    params <- mluPasteArg(inputModelFile, "character", params)
    params <- mluPasteArg(loadTransforms, "logical", params)
    params <- mluPasteArg(randomSeed, "integer", params)
    params <- mluPasteArg(parallel, "integer", params)
    params <- mluPasteArg(transform.with.tag, "list", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    active.calls <- sapply(sys.calls(), function(acall) as.character(acall[[1]]))
    if (!nzchar(commandContext) && length(active.calls) > 1) {
        chainIds <- grep("chain", active.calls, ignore.case = TRUE)
        sweepIds <- grep("sweep", active.calls, ignore.case = TRUE)
        if (length(chainIds) > 0 && length(sweepIds) == 0)
            commandContext <- "chain"
        else if (length(sweepIds) > 0 && length(chainIds) ==
            0)
            commandContext <- "sweep"
        else if (length(sweepIds) > 0 && length(chainIds) > 0)
            commandContext <- ifelse(max(chainIds) > max(sweepIds),
                "chain", "sweep")
    }
    params <- switch(commandContext, chain = sprintf("Score{%s}",
        params), sweep = sprintf("{Score%s}", params), sprintf("Score%s",
        params))
    if (commandContext == "")
        mamlRun(params)
    else return(structure(params, class = c("Score", "Command", "maml", "character")))
  }
```

`tlcTrain` *Command: 'Train'*

Description

Trains a predictor.

Usage

```
tlcTrain(customColumn.with.tag = "", normalize = "Auto",
  trainer = "AveragedPerceptron", validationFile = "",
  calibrator = "sigmoid", randomSeed = NULL, parallel = NULL,
  transform.with.tag = list(), ..., commandContext = "")
```

Arguments

`customColumn.with.tag`	character: <string>. Columns with custom kinds declared through key assignments, e.g., col[Kind]=Name to assign column named 'Name' kind 'Kind' (short form col)
`normalize`	character: [No|Warn|Auto|Yes]. Normalize option for the feature column Default value:'Auto' (short form norm)
`trainer`	list: <name><options>. Trainer to use Default value:'AveragedPerceptron' (short form tr)

`validationFile`	character: <string>. The validation data file (short form valid)
`calibrator`	list: <name><options>. Output calibrator Default value:'PlattCalibration' (short form cali)
`randomSeed`	integer: <int>. Random seed (short form seed)
`parallel`	integer: <int>. Desired degree of parallelism in the data pipeline (short form n)
`transform.with.tag`	list: <name><options>. Transform (short form xf)
`commandContext`	character: [chain|sweep|]. The compute context of the command. Default value:"
`...`	: . hidden arguments

Value

the output of the TLC Command: Train (Command).

Note

args with tag (customColumn.with.tag, transform.with.tag) can be specified as: structure('...', tag = '...').

Author(s)

Microsoft Corporation

References

https://microsoft.sharepoint.com/teams/TLC

Examples

```
##---- Should be DIRECTLY executable !! ----
##-- ==>  Define data, use random,
##--or do  help(data=index)  for the standard data sets.

## The function is currently defined as
function (featureColumn = "Features", labelColumn = "Label",
    weightColumn = "Weight", groupColumn = "GroupId", nameColumn = "Name",
   customColumn.with.tag = "", normalizeFeatures = "Auto", trainer = "AveragedPerceptron",
    validationFile = "", cacheData = FALSE, calibrator = "PlattCalibration",
    maxCalibrationExamples = 1e+09, continueTrain = FALSE, loader = list(),
    dataFile = "", outputModelFile = "", inputModelFile = "",
    loadTransforms = FALSE, randomSeed = NULL, parallel = NULL,
    transform.with.tag = list(), ..., commandContext = "")
{
    params <- character()
    params <- mluPasteArg(featureColumn, "character", params)
    params <- mluPasteArg(labelColumn, "character", params)
    params <- mluPasteArg(weightColumn, "character", params)
    params <- mluPasteArg(groupColumn, "character", params)
    params <- mluPasteArg(nameColumn, "character", params)
    params <- mluPasteArg(customColumn.with.tag, "character", params)
    params <- mluPasteArg(normalizeFeatures, "character", params)
    params <- mluPasteArg(trainer, "list", params)
    params <- mluPasteArg(validationFile, "character", params)
    params <- mluPasteArg(cacheData, "logical", params)
    params <- mluPasteArg(calibrator, "list", params)
    params <- mluPasteArg(maxCalibrationExamples, "integer", params)
    params <- mluPasteArg(continueTrain, "logical", params)
    params <- mluPasteArg(loader, "list", params)
    params <- mluPasteArg(dataFile, "character", params)
    params <- mluPasteArg(outputModelFile, "character", params)
    params <- mluPasteArg(inputModelFile, "character", params)
    params <- mluPasteArg(loadTransforms, "logical", params)
    params <- mluPasteArg(randomSeed, "integer", params)
    params <- mluPasteArg(parallel, "integer", params)
    params <- mluPasteArg(transform.with.tag, "list", params)
    dotargs <- list(...)
    for (arg in names(dotargs)) {
        params <- mluPasteArg(dotargs[[arg]], "", params, arg)
    }
    active.calls <- sapply(sys.calls(), function(acall) as.character(acall[[1]]))
    if (!nzchar(commandContext) && length(active.calls) > 1) {
        chainIds <- grep("chain", active.calls, ignore.case = TRUE)
        sweepIds <- grep("sweep", active.calls, ignore.case = TRUE)
        if (length(chainIds) > 0 && length(sweepIds) == 0)
            commandContext <- "chain"
        else if (length(sweepIds) > 0 && length(chainIds) ==
            0)
            commandContext <- "sweep"
        else if (length(sweepIds) > 0 && length(chainIds) > 0)
            commandContext <- ifelse(max(chainIds) > max(sweepIds),
                "chain", "sweep")
    }
    params <- switch(commandContext, chain = sprintf("Train{%s}",
        params), sweep = sprintf("{Train%s}", params), sprintf("Train%s",
        params))
    if (commandContext == "")
        mamlRun(params)
    else return(structure(params, class = c("Train", "Command", "maml", "character")))
  }
```

Index