Package 'superml'

January 8, 2019

Type Package

Title Build Machine Learning Models Like Using Python's Scikit-Learn Library in R

Version 0.2.0

Maintainer Manish Saraswat <manish06saraswat@gmail.com>

Description The idea is to provide a standard interface to users who use both R and Python for building machine learning models. This package provides a scikit-learn's fit, predict interface to train machine learning models in R.

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

URL https://github.com/saraswatmks/superml

BugReports https://github.com/saraswatmks/superml/issues

Depends R(>= 3.4), R6(>= 2.2)

Imports data.table(>= 1.10), assertthat(>= 0.2), Metrics(>= 0.1), xgboost(>= 0.6), glmnet(>= 2.0), parallel, kableExtra, tm(>= 0.7), naivebayes(>= 0.9), ClusterR(>= 1.1), FNN(>= 1.1), liquidSVM(>= 1.2), ranger(>= 0.10), caret(>= 6.0), doParallel(>= 1.0)

Suggests knitr, rlang, testthat, rmarkdown

RoxygenNote 6.1.1

VignetteBuilder knitr

NeedsCompilation no

Author Manish Saraswat [aut, cre]

Repository CRAN

Date/Publication 2019-01-07 23:00:06 UTC

R topics documented:

`bm25` *Best Matching(BM25)*

Description

BM25 stands for Best Matching 25. It is widely using for ranking documents and a preferred method than TF*IDF scores. It is used to find the similar documents from a corpus, given a new document. It is popularly used in information retrieval systems. This implementation uses multiple cores for faster and parallel computation.

Usage

```
bm25
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bm25 = bm25$new(corpus, n_cores)
bm25$most_similar(input_document, topn)
bm25$compute(input_document)
```

Methods

`$new()` Initialise the instance of the class. Here you pass the complete corpus of the documents

`$most_similar()` it returns the topn most similar documents from the corpus

`$compute()` it returns a similarity score for all the documents in the corpus, given a sentence

Arguments

corpus a list containing sentences

use_parallel boolean value used to activate parallel computation, defaults to FALSE

Examples

```
example <- c('white audi 2.5 car','black shoes from office',
             'new mobile iphone 7','audi tyres audi a3',
             'nice audi bmw toyota corolla')
get_bm <- bm25$new(example, use_parallel=FALSE)
input_document <- c('white toyota corolla')
get_bm$most_similar(document = input_document, topn = 2)
```

---

cla_train *cla_train*

Description

Training Dataset used for classification examples. This is classic titanic dataset used to predict if a passenger will survive or not in titanic ship disaster.

Usage

```
cla_train
```

Format

A data frame with 12 variables:

`PassengerId` unique ID of the passenger

`Survived` survival of the passenger

`Pclass` ticket class

`Name` name of the passenger

`Sex` sex of the passenger

`Age` Age of the passenger

`SibSp` number of siblings/spouse aboard the titanic

`Parch` number of parents / child aboard the titanic

`Ticket` ticket number

`Fare` passenger fare

`Cabin` cabin number

`Embarked` port of embarkation

Source

https://www.kaggle.com/c/titanic/data

Counter *Calculate count of values in a list or vector*

Description

Handy function to calculate count of values given in a list or vector

Usage

```
Counter(data, sort = TRUE, decreasing = FALSE)
```

Arguments

data	should be a vector or list of input values
sort	a logical value, to sort the result or not
decreasing	a logical value, the order of sorting to be followed

Value

count of values in a list

Examples

```
d <- list(c('i','am','bad'),c('you','are','also','bad'))
counts <- Counter(d, sort=TRUE, decreasing=TRUE)
```

CountVectorizer *Count Vectorizer*

Description

Creates CountVectorizer Model. Given a list of text, it generates a bag of words model and returns a data frame consisting of BOW features.

Usage

```
CountVectorizer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bst = CountVectorizer$new(min_df=1, max_df=1, max_features=1)
bst$fit(sentences)
bst$fit_transform(sentences)
bst$transform(sentences)
```

Methods

`$new()` Initialise the instance of the vectorizer

`$fit()` creates a memory of bag of words

`$transform()` based on encodings learned in `fit` method, return a bag of words matrix

`$fit_transform()` simultaneouly fits and transform words and returns bag of words of matrix

Examples

```
df <- data.frame(sents = c('i am alone in dark.','mother_mary a lot',
                           'alone in the dark?',
                           'many mothers in the lot....'))

# fits and transforms on the entire data in one go
bw <- CountVectorizer$new(min_df = 0.3)
tf_features <- bw$fit_transform(df$sents)

# fit on entire data and do transformation in train and test
bw <- CountVectorizer$new()
bw$fit(df$sents)
tf_features <- bw$transform(df$sents)
```

GridSearchCV *Grid Search CV*

Description

Grid search CV is used to train a machine learning model with multiple combinations of training hyper parameters and finds the best combination of parameters which optimizes the evaluation metric. It creates an exhaustive set of hyperparameter combinations and train model on each combination.

Usage

```
GridSearchCV
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
gst = GridSearchTrainer$new(trainer, parameters, n_folds, scoring)
gst$fit(X, y)
gst$best_iteration(metric)
```

Methods

`$new()` Initialises an instance of grid search cv

`$fit()` fit model to an input train data and trains the model.

`$best_iteration()` returns best iteration based on a given metric. By default, uses the first scoring metric

Arguments

trainer superml trainer object, could be either XGBTrainer, RFTrainer, NBTrainer etc.

parameters list containing parameters

n_folds number of folds to use to split the train data

scoring scoring metric used to evaluate the best model, multiple values can be provided. currently supports: auc, accuracy, mse, rmse, logloss, mae, f1, precision, recall

Examples

```
rf <- RFTrainer$new()
gst <-GridSearchCV$new(trainer = rf,
                           parameters = list(n_estimators = c(100),
                                             max_depth = c(5,2,10)),
                                             n_folds = 3,
                                             scoring = c('accuracy','auc'))
data("iris")
gst$fit(iris, "Species")
gst$best_iteration()
```

---

kFoldMean *kFoldMean Calculator*

Description

Calculates out-of-fold mean features (also known as target encoding) for train and test data. This strategy is widely used to avoid overfitting or causing leakage while creating features using the target variable. This method is experimental. If the results you get are unexpected, please report them in github issues.

Usage

```
kFoldMean(train_df, test_df, colname, target, n_fold = 5, seed = 42)
```

Arguments

train_df	train dataset
test_df	test dataset
colname	name of categorical column
target	the target or dependent variable, should be a string.
n_fold	the number of folds to use for doing kfold computation, default=5
seed	the seed value, to ensure reproducibility, it could be any positive value, default=42

Value

a train and test data table with out-of-fold mean value of the target for the given categorical variable

Examples

```
train <- data.frame(region=c('del','csk','rcb','del','csk','pune','guj','del'),
                    win = c(0,1,1,0,0,0,0,1))
test <- data.frame(region=c('rcb','csk','rcb','del','guj','pune','csk','kol'))
train_result <- kFoldMean(train_df = train,
                          test_df = test,
                          colname = 'region',
                          target = 'win',
                          seed = 1220)$train

test_result <- kFoldMean(train_df = train,
                         test_df = test,
                         colname = 'region',
                         target = 'win',
                         seed = 1220)$test
```

KMeansTrainer *K-Means Trainer*

Description

Trains a unsupervised K-Means clustering algorithm. It borrows mini-batch k-means function from ClusterR package written in c++, hence it is quite fast.

Usage

```
KMeansTrainer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
kmt = KMeansTrainer$new(clusters, batch_size = 10, num_init=1, max_iters=100, init_fraction=1,
                    initializer = "kmeans++", early_stop_iter = 10, verbose=FALSE, centroids=NULL,
                      tol = 1e-04, tol_optimal_init=0.3, seed=1, max_clusters=NA)
bst$fit(X_train, y_train=NULL)
prediction <- bst$predict(X_test)
```

Methods

`$new()` Initialises an instance of k-means model

`$fit()` fit model to an input train data

`$predict()` returns cluster predictions for each row of given data

Arguments

params for explanation on parameters, please refer to the documentation of MiniBatchKMeans function in clusterR package https://CRAN.R-project.org/package=ClusterR

find_optimal Used to find the optimal number of cluster during `fit` method. To use this, make sure the value for max_clusters > 0.

Examples

```
data <- rbind(replicate(20, rnorm(1e4, 2)),
             replicate(20, rnorm(1e4, -1)),
             replicate(20, rnorm(1e4, 5)))
km_model <- KMeansTrainer$new(clusters=2, batch_size=30, max_clusters=6)
km_model$fit(data, find_optimal = FALSE)
predictions <- km_model$predict(data)
```

`KNNTrainer` *K Nearest Neighbours Trainer*

Description

Trains a k nearest neighbour model using fast search algorithms. KNN is a supervised learning algorithm which is used for both regression and classification problems.

Usage

```
KNNTrainer
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bst = KNNTrainer$new(k=1, prob=FALSE, algorithm=NULL, type="class")
bst$fit(X_train, X_test, "target")
bst$predict(type)
```

Methods

`$new()` Initialise the instance of the trainer

`$fit()` trains the knn model and stores the test prediction

`$predict()` returns predictions

Arguments

k number of neighbours to predict

prob if probability should be computed, default=FALSE

algorithm algorithm used to train the model, possible values are 'kd_tree','cover_tree','brute'

type type of problem to solve i.e. regression or classification, possible values are 'reg' or 'class'

Examples

```
data("iris")

iris$Species <- as.integer(as.factor(iris$Species))

xtrain <- iris[1:100,]
xtest <- iris[101:150,]

bst <- KNNTrainer$new(k=3, prob=TRUE, type="class")
bst$fit(xtrain, xtest, 'Species')
pred <- bst$predict(type="raw")
```

LabelEncoder *Label Encoder*

Description

Encodes and decodes categorical variables into integer values and vice versa. This is a commonly performed task in data preparation during model training, because all machine learning models require the data to be encoded into numerical format. It takes a vector of character or factor values and encodes them into numeric.

Usage

```
LabelEncoder
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
lbl = LabelEncoder$new()
lbl$fit(x)
lbl$fit_transform(x)
lbl$transform(x)
```

Methods

`$new()` Initialise the instance of the encoder

`$fit()` creates a memory of encodings but doesn't return anything

`$transform()` based on encodings learned in `fit` method is applies the transformation

`$fit_transform()` encodes the data and keep a memory of encodings simultaneously

`$inverse_transform()` encodes the data and keep a memory of encodings simultaneously

Arguments

data a vector or list containing the character / factor values

Examples

```
data_ex <- data.frame(Score = c(10,20,30,4), Name=c('Ao','Bo','Bo','Co'))
lbl <- LabelEncoder$new()
data_ex$Name <- lbl$fit_transform(data_ex$Name)
decode_names <- lbl$inverse_transform(data_ex$Name)
```

LMTrainer *Linear Models Trainer*

Description

Trains linear models such as Logistic, Lasso or Ridge regression model. It is built on glmnet R package. This class provides fit, predict, cross valdidation functions.

Usage

```
LMTrainer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bst = LMTrainer$new(family, weights, alpha, lambda=100, standardize.response=FALSE)
bst$fit(X_train, "target")
prediction <- bst$predict(X_test)
bst$cv_model(X_train, "target", nfolds=4, parallel=TRUE)
cv_prediction <- bst$cv_predict(X_test)
```

Methods

`$new()` Initialises an instance of random forest model

`$fit()` fit model to an input train data (data frame or data table) and trains the model.

`$predict()` returns predictions by fitting the trained model on test data.

`$cv_model()` Using k-fold cross validation technique, finds the best value of lambda. type.measure is the loss to use for cross validation.

`$cv_predict()` Using the best value of lambda, makes predictions on the test data

`$get_importance()` Returns a matrix of feature coefficients as generated by Lasso

Arguments

family type of regression to perform, values can be "gaussian" ,"binomial", "multinomial","mgaussian"

weights observation weights. Can be total counts if responses are proportion matrices. Default is 1 for each observation

alpha The elasticnet mixing parameter, alpha=1 is the lasso penalty, and alpha=0 the ridge penalty.

nlambda the number of lambda values - default is 100

standardize.response normalise the dependent variable between 0 and 1, default = FALSE

Examples

```
LINK <- "http://archive.ics.uci.edu/ml/machine-learning-databases/housing/housing.data"
housing <- read.table(LINK)
names <- c("CRIM","ZN","INDUS","CHAS","NOX","RM","AGE","DIS",
           "RAD","TAX","PTRATIO","B","LSTAT","MEDV")
names(housing)  <-  names
lf <- LMTrainer$new(family = 'gaussian', alpha=1)
lf$fit(X = housing, y = 'MEDV')
predictions <- lf$predict(df = housing)


# cross validation model
lf$cv_model(X = housing, y = 'MEDV', nfolds = 5, parallel = FALSE)
predictions <- lf$cv_predict(df = housing)
coefs <- lf$get_importance()
```

NBTrainer *Naive Bayes Trainer*

Description

Trains a naive bayes model. It is built on top high performance naivebayes R package.

Usage

```
NBTrainer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
nbt = NBTrainer$new(prior=NULL, laplace=0, usekernel=FALSE)
nbt$fit(X_train, "target")
prediction <- bst$predict(X_test)
```

Methods

`$new()` Initialises an instance of naive bayes model

`$fit()` fits model to an input train data and trains the model.

`$predict()` returns predictions by fitting the trained model on test data.

Arguments

prior for detailed explanation of parameters, check: https://cran.r-project.org/package=naivebayes

prior numeric vector with prior probabilities. vector with prior probabilities of the classes. If unspecified, the class proportions for the training set are used. If present, the probabilities should be specified in the order of the factor levels.

laplace value used for Laplace smoothing. Defaults to 0 (no Laplace smoothing)

usekernel if TRUE, density is used to estimate the densities of metric predictors

Examples

```
data(iris)
nb <- NBTrainer$new()
nb$fit(iris, 'Species')
y <- nb$predict(iris)
```

RandomSearchCV *Random Search CV*

Description

Given a set of hyper parameters, random search trainer provides a faster way of hyper parameter tuning. Here, the number of models to be trained can be defined by the user.

Usage

```
RandomSearchCV
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
rst = RandomSearchTrainer$new(trainer, parameters, n_folds, scoring, n_iter)
rst$fit(X_train, "target")
rst$best_iteration(metric)
```

Methods

`$new()` Initialises an instance of random search cv

`$fit()` fit model to an input train data and trains the model.

`$best_iteration()` returns best iteration based on a given metric. By default, uses the first scoring metric

Arguments

trainer superml trainer object, must be either XGBTrainer, LMTrainer, RFTrainer, NBTrainer

parameters list containing parameters

n_folds number of folds to use to split the train data

scoring scoring metric used to evaluate the best model, multiple values can be provided. currently supports: auc, accuracy, mse, rmse, logloss, mae, f1, precision, recall

n_iter number of models to be trained

Examples

```
rf <- RFTrainer$new()
rst <-RandomSearchCV$new(trainer = rf,
                         parameters = list(n_estimators = c(100,500),
                         max_depth = c(5,2,10,14)),
                         n_folds = 3,
                         scoring = c('accuracy','auc'),
                         n_iter = 4)
data("iris")
rst$fit(iris, "Species")
rst$best_iteration()
```

`reg_train` *reg_train*

Description

Training Dataset used for regression examples. In this data set, we have to predict the sale price of the houses.

Usage

```
reg_train
```

Format

An object of class `data.table` (inherits from `data.frame`) with 1460 rows and 81 columns.

`RFTrainer` *Random Forest Trainer*

Description

Trains a Random Forest model. A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. This implementation uses ranger R package which provides faster model training.

Usage

```
RFTrainer
```

Format

`R6Class` object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bst = RandomForestTrainer$new(n_estimators=100, max_features="auto", max_depth=5, min_node_size=1,
                          criterion, classification=1, class_weights, verbose=TRUE,
                               seed=42, always_split)
bst$fit(X_train, "target")
prediction <- bst$predict(X_test)
```

Methods

`$new()` Initialises an instance of random forest model

`$fit()` fit model to an input train data and trains the model.

`$predict()` returns predictions by fitting the trained model on test data.

`$get_importance()` Get feature importance from the model

Arguments

n_estimators the number of trees in the forest, default= 100

max_features the number of features to consider when looking for the best split. Possible values are `auto(default)` takes sqrt(num_of_features), `sqrt` same as auto, `log` takes log(num_of_features), `none` takes all features

max_depth the maximum depth of each tree

min_node_size the minumum number of samples required to split an internal node

criterion the function to measure the quality of split. For classification, `gini` is used which is a measure of gini index. For regression, the `variance` of responses is used.

classification whether to train for classification (1) or regression (0)

class_weights weights associated with the classes for sampling of training observation

verbose show computation status and estimated runtime

always_split vector of feature names to be always used for splitting

seed seed value

importance Variable importance mode, one of 'none', 'impurity', 'impurity_corrected', 'permutation'. The 'impurity' measure is the Gini index for classification, the variance of the responses for regression. Defaults to "impurity"

Examples

```
data("iris")
bst <- RFTrainer$new(n_estimators=50,
                     max_depth=4,
                     classification=1,
                     seed=42,
                     verbose=TRUE)
bst$fit(iris, 'Species')
predictions <- bst$predict(iris)
bst$get_importance()
```

smoothMean *smoothMean Calculator*

Description

Calculates target encodings using a smoothing parameter and count of categorical variables. This approach is more robust to possibility of leakage and avoid overfitting.

Usage

```
smoothMean(train_df, test_df, colname, target, min_samples_leaf = 1,
  smoothing = 1, noise_level = 0)
```

Arguments

train_df	train dataset
test_df	test dataset
colname	name of categorical column
target	name of target column
min_samples_leaf	minimum samples to take category average into account
smoothing	smoothing effect to balance categorical average vs prior
noise_level	random noise to add, optional

Value

a train and test data table with mean encodings of the target for the given categorical variable

Examples

```
train <- data.frame(region=c('del','csk','rcb','del','csk','pune','guj','del'),
                    win = c(0,1,1,0,0,1,0,1))
test <- data.frame(region=c('rcb','csk','rcb','del','guj','pune','csk','kol'))

# calculate encodings
all_means <- smoothMean(train_df = train,
                        test_df = test,
                        colname = 'region',
                        target = 'win')
train_mean <- all_means$train
test_mean <- all_means$test
```

SVMTrainer *Support Vector Machines Trainer*

Description

Trains a support vector machine (svm) model. It is based on the magnificently fast speed liquidSVM R package. It provides a more unified interface over the package retaining all its functionality.

The model is intelligently trained with a default set of hyper parameters. Also, there are inbuilt grid setups which can be easily initialised. It has capability to support batch processing of data to avoid memory errors. It supports binary classification, multi classification, regression models

Usage

```
SVMTrainer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
svm = SVMTrainer$new(type=NULL,scale=TRUE, gammas=NULL, lambdas=NULL, c_values=NULL,predict.prob=FAL
                     verbose=NULL, ncores=NULL, partition_choice=0,
                     seed=-1, grid_choice=NULL, useCells=FALSE, mc_type=NULL,
                     adaptivity_control=0)
svm$fit(X_train, y_train)
prediction <- svm$predict(X_test)
```

Methods

`$new()` Initialises an instance of svm model

`$fit()` fits model to an input train data and trains the model.

`$predict()` returns predictions by fitting the trained model on test data.

Arguments

params for detailed explanation on parameters, refer to original documentation https://cran.r-project.org/package=liquidSVM

type type of model to train, possible values: "bc" = binary classification, "mc" = multiclassification, "ls" = least square regression, "qt" = quantile regression

scale normalises the feature between 0 and 1, default = TRUE

gammas bandwidth of the kernel, default value is chosen from a list of gamma values generated internally

lambdas regularization parameter

c_values cost parameter

predict.prob If TRUE then final prediction is probability else labels. This also restricts the choices of mc_type to c("OvA_ls","AvA_ls").

verbose display the progress to standard output, possible values are 0, 1

ncores number of cores to use for parallel processing, possible values are 0 (default), -1

partition_choice optimization parameter to train on large data sets, possible value are: 0 (disables partitioning) , 6 (high speed), 5 (best error)

seed random seed, default = -1

grid_choice internal grid used for convenient hyperparameter tuning of gammas, lambdas, possible values are: 0,1,2,-1,-2

useCells activates batch processing, set it to TRUE in case of out of memory errors

mc_type configure multiclassification variant like OnevsAll, AllvsAll, possible values are: "AvA_hinge", "OvA_ls", "OvA_hinge", "AvA_ls"

quantile do quantile regression, default=FALSE

weights weights to be used in quantile regression, default is c(0.05, 0.1, 0.5, 0.9, 0.95)

Examples

```
data(iris)
## Multiclassification
svm <- SVMTrainer$new(type="mc")
svm$fit(iris, "Species")
p <- svm$predict(iris)

## Least Squares
svm <- SVMTrainer$new(type="ls")
svm$fit(trees, "Height")
p <- svm$predict(trees)

## Quantile regression
svm <- SVMTrainer$new(type="qt")
svm$fit(trees,"Height")
p <- svm$predict(trees)
```

TfIdfVectorizer *TfIDF(Term Frequency Inverse Document Frequency) Vectorizer*

Description

Provides an easy way to create tf-idf matrix of features in R. It consists of fit, transform methods (similar to sklearn) to generate tf-idf features.

Usage

```
TfIdfVectorizer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
tf_object = TfIdfVectorizer$new(max_df=1, min_df=1, max_features=1, smooth_idf=TRUE)
tf_object$fit(sentences)
tf_matrix = tf_object$transform(sentences)
tf_matrix = tf_object$fit_transform(sentences) ## alternate
```

Methods

`$new()` Initialise the instance of the vectorizer

`$fit()` creates a memory of count vectorizers but doesn't return anything

`$transform()` based on encodings learned in `fit` method, returns the tf-idf matrix

`$fit_transform()` returns tf-idf matrix

Examples

```
df <- data.frame(sents = c('i am alone in dark.',
                           'mother_mary a lot',
                           'alone in the dark?',
                           'many mothers in the lot....'))
tf <- TfIdfVectorizer$new(smooth_idf = TRUE, min_df = 0.3)
tf_features <- tf$fit_transform(df$sents)
```

XGBTrainer *Extreme Gradient Boosting Trainer*

Description

Trains a Extreme Gradient Boosting Model. XGBoost belongs to a family of boosting algorithms that creates an ensemble of weak learner to learn about data.

Usage

```
XGBTrainer
```

Format

R6Class object.

Usage

For usage details see **Methods, Arguments and Examples** sections.

```
bst = XGBTrainer$new(booster, objective, nthread, silent=0, n_estimators=100, learning_rate=0.3,
                 gamma=0, max_depth=6, min_child_weight=1, subsample=1, colsample_bytree=1,
                 lambda=1, alpha = 0, eval_metric, print_every = 50, feval, early_stopping, maximize,
                 custom_objective, save_period, save_name, xgb_model, callbacks, verbose,
                     num_class, weight, na_missing)
bst$fit(X_train, "target", valid=NULL)
tfidf$predict(X_test)
```

Methods

`$new()` Initialises an instance of xgboost model

`$fit()` fits model to an input train data using given parametes.

`$cross_val()` performs cross validation on train data

`$predict()` returns predictions by fitting the trained model on test data.

Arguments

params the detailed version of these arguments can found in xgboost documentation http://xgboost.readthedocs.io/en/latest/parameter.html

booster the trainer type, the values are `gbtree(default)`, `gblinear`, `dart:gbtree`

objective specify the learning task. Check the link above for all possible values.

max_features the number of features to consider when looking for the best split Possible values are `auto`,`sqrt`,`log`,`None`

n_thread number of parallel threads used to run, default is to run using all threads available

silent 0 means printing running messages, 1 means silent mode

n_estimators number of trees to grow, default = 100

learning_rate Step size shrinkage used in update to prevents overfitting. Lower the learning rate, more time it takes in training, value lies between between 0 and 1. Default = 0.3

max_depth the maximum depth of each tree, default = 6

nfold set number of folds for cross validation

stratified if stratified sampling to be done or not, default is TRUE

folds the list of CV folds' indices - either those passed through the folds parameter or randomly generated.

gamma Minimum loss reduction required to make a further partition on a leaf node of the tree. The larger gamma is, the more conservative the algorithm will be. Value lies between 0 and infinity, Default = 0

min_child_weight Minimum sum of instance weight (hessian) needed in a child. If the tree partition step results in a leaf node with the sum of instance weight less than min_child_weight, then the building process will give up further partitioning. In linear regression task, this simply corresponds to minimum number of instances needed to be in each node. The larger min_child_weight is, the more conservative the algorithm will be. Value lies between 0 and infinity. Default = 1

subsample Subsample ratio of the training instances. Setting it to 0.5 means that XGBoost would randomly sample half of the training data prior to growing trees. and this will prevent overfitting. Subsampling will occur once in every boosting iteration. Value lies between 0 and 1. Default = 1

colsample_bytree Subsample ratio of columns when constructing each tree. Subsampling will occur once in every boosting iteration. Value lies between 0 and 1. Default = 1

lambda L2 regularization term on weights. Increasing this value will make model more conservative. Default = 1

alpha L1 regularization term on weights. Increasing this value will make model more conservative. Default = 0

eval_metric Evaluation metrics for validation data, a default metric will be assigned according to objective

print_every print training log after n iterations. Default = 50

feval custom evaluation function

early_stopping Used to prevent overfitting, stops model training after this number of iterations if there is no improvement seen

maximize If feval and early_stopping_rounds are set, then this parameter must be set as well. When it is TRUE, it means the larger the evaluation score the better.

custom_objective custom objective function

save_period when it is non-NULL, model is saved to disk after every save_period rounds, 0 means save at the end.

save_name the name or path for periodically saved model file.

xgb_model a previously built model to continue the training from. Could be either an object of class xgb.Booster, or its raw data, or the name of a file with a previously saved model.

callbacks a list of callback functions to perform various task during boosting. See callbacks. Some of the callbacks are automatically created depending on the parameters' values. User can provide either existing or their own callback methods in order to customize the training process.

verbose If 0, xgboost will stay silent. If 1, xgboost will print information of performance. If 2, xgboost will print some additional information. Setting verbose > 0 automatically engages the cb.evaluation.log and cb.print.evaluation callback functions.

watchlist what information should be printed when verbose=1 or verbose=2. Watchlist is used to specify validation set monitoring during training. For example user can specify watchlist=list(validation1=mat1, validation2=mat2) to watch the performance of each round's model on mat1 and mat2

num_class set number of classes in case of multiclassification problem

weight a vector indicating the weight for each row of the input.

na_missing by default is set to NA, which means that NA values should be considered as 'missing' by the algorithm. Sometimes, 0 or other extreme value might be used to represent missing values. This parameter is only used when input is a dense matrix.

Examples

```
library(data.table)
df <- copy(iris)

# convert characters/factors to numeric
df$Species <- as.numeric(as.factor(df$Species))-1

# initialise model
xgb <- XGBTrainer$new(objective = 'multi:softmax',
                      maximize = FALSE,
                      eval_metric = 'merror',
                      num_class=3,
                      n_estimators = 2)
xgb$fit(df, 'Species')

# do cross validation to find optimal value for n_estimators
xgb$cross_val(X = df, y = 'Species',nfolds = 3, stratified = TRUE,
              early_stopping = 1)
best_iter <- xgb$cv_model$best_iteration
xgb$show_importance()

# make predictions
preds <- xgb$predict(as.matrix(iris[,1:4]))
preds
```

Index