Package 'ssr'

October 14, 2022

Type Package

Title Semi-Supervised Regression Methods

Version 0.1.1

Description An implementation of semi-supervised regression methods including self-learning and co-training by committee based on Hady, M. F. A., Schwenker, F., & Palm, G. (2009) <doi:10.1007/978-3-642-04274-4_13>. Users can define which set of regressors to use as base models from the 'caret' package, other packages, or custom functions.

Depends R (>= 3.6.0)

License GPL-3

Encoding UTF-8

URL https://github.com/enriquegit/ssr

BugReports https://github.com/enriquegit/ssr/issues

LazyData true

Imports caret, e1071

RoxygenNote 6.1.1

Suggests knitr, rmarkdown, tgp

VignetteBuilder knitr

NeedsCompilation no

Author Enrique Garcia-Ceja [aut, cre] (<https://orcid.org/0000-0001-6864-8557>)

Maintainer Enrique Garcia-Ceja <e.g.mx@ieee.org>

Repository CRAN

Date/Publication 2019-09-02 15:40:03 UTC

R topics documented:

`friedman1` *friedman1 dataset.*

Description

A dataset generated with `friedman.1.data` from 'tgp' package. The friedman1 data set is commonly used to test semi-supervised regression methods.

Usage

```
friedman1
```

Format

A data frame with 1000 rows and 11 numeric variables. All variables were scaled to 0-1. X1 to X10 are the input variables and Ytrue is the output variable.

`plot.ssr` *Plots a ssr object*

Description

Plots the results of a fitted ssr object if a testset was provided when fitting the model.

Usage

```
## S3 method for class 'ssr'
plot(x, metric = "rmse", ptype = 1, ...)
```

Arguments

`x`	a fitted object of class "ssr".
`metric`	the type of metric to be plotted ("rmse", "mae", "cor"), defaults to "rmse". "cor" is for pearson correlation.
`ptype`	an integer specifying the type of plot. The default 1, plots the performance metric of the fitted model. Any value different of 1, plots the performance metric of the individual regressors used to build the model.
`...`	additional arguments to be passed to the plot function.

Details

This function generates performance plots to quickly inspect the results of the fitted model. The fitted model contains all the necessary data so the user can create custom plots, if required.

Value

a NULL invisible object.

Examples

```
dataset <- friedman1 # Load dataset.

set.seed(1234)

# Prepare the data.
split1 <- split_train_test(dataset, pctTrain = 70)
split2 <- split_train_test(split1$trainset, pctTrain = 5)
L <- split2$trainset
U <- split2$testset[, -11]
testset <- split1$testset
regressors <- list(knn = caret::knnreg)
model <- ssr("Ytrue ~ .", L, U, regressors = regressors, testdata = testset, maxits = 10)

# Plot the RMSE of the fitted model.
plot(model, metric = "rmse", ptype = 1)

# Plot the MAE.
plot(model, metric = "mae", ptype = 1)
```

---

predict.ssr *Predictions from a fitted ssr object*

Description

Returns a vector of predicted responses from the fitted ssr object.

Usage

```
## S3 method for class 'ssr'
predict(object, newdata, ...)
```

Arguments

object	fitted object of class ssr.
newdata	data frame with the input variables from which the response variable is to be predicted.
...	additional arguments (not used)

Value

A numeric vector with the predictions for each row of the input data frame.

Examples

```
dataset <- friedman1 # Load friedman1 dataset.

set.seed(1234)

# Split the dataset into 70% for training and 30% for testing.
split1 <- split_train_test(dataset, pctTrain = 70)

# Choose 5% of the train set as the labeled set L and the remaining will be the unlabeled set U.
split2 <- split_train_test(split1$trainset, pctTrain = 5)

L <- split2$trainset

U <- split2$testset[, -11] # Remove the labels.

testset <- split1$testset

regressors <- list(knn = caret::knnreg)

model <- ssr("Ytrue ~ .", L, U, regressors = regressors, testdata = testset, maxits = 10)

# Plot RMSE.
plot(model)

# Get the predictions on the testset.
predictions <- predict(model, testset)

# Calculate RMSE on the test set.
sqrt(mean((predictions - testset$Ytrue)^2))
```

---

split_train_test *Splits a data frame into train and test sets.*

Description

Utility function to randomly split a data frame into train and test sets.

Usage

```
split_train_test(df, pctTrain)
```

Arguments

df	a data frame.
pctTrain	numeric value that specifies the percentage of rows to be included in the train set. The remaining rows are added to the test set.

Value

a list with the first element being the train set and the second element the test set.

Examples

```
set.seed(1234)

dataset <- friedman1

nrow(dataset) # print number of rows

split1 <- split_train_test(dataset, pctTrain = 70) # select 70% for training

nrow(split1$trainset) # number of rows of the train set

nrow(split1$testset) # number of rows of the test set

head(split1$trainset) # display first rows of train set
```

---

ssr *Fits a semi-supervised regression model*

Description

This function implements the *co-training by committee* and *self-learning* semi-supervised regression algorithms with a set of *n* base regressor(s) specified by the user. When only one model is present in the list of regressors, self-learning is performed.

Usage

```
ssr(theFormula, L, U, regressors = list(lm = lm, knn = caret::knnreg),
  regressors.params = NULL, pool.size = 20, gr = 1, maxits = 20,
  testdata = NULL, shuffle = TRUE, verbose = TRUE,
  plotmetrics = FALSE, U.y = NULL)
```

Arguments

`theFormula`	a `formula` that specifies the response and the predictor variables. Two formats are supported: `"Y ~ ."` and `"Y ~ var1 + var2 + ... + varn"`.
`L`	a data frame that contains the initial labeled training set.
`U`	a data frame that contains the unlabeled data. If the provided data frame has the response variable as one of its columns, it will be discarded.
`regressors`	a list of custom functions and/or strings naming the regression models to be used. The strings must contain a valid name of a regression model from the 'caret' package. The list of available regression models from the 'caret' package can be found here. Functions must be named, e.g., `list(linearModel=lm)`.

List names for models defined with strings are optional. A list can contain both, strings and functions: `list("kknn", linearModel=lm)`. For better performance in time, it is recommended to pass functions directly rather than using 'caret' strings since 'caret' does additional preprocessing when training models. Examples can be found in the vignettes.

`regressors.params`
: a list of lists that specifies the parameters for each custom function. For 'caret' models specified as strings in `regressors`, parameters cannot be passed, use `NULL` instead. The parameters are specified with a named list. For example, if `regressors = list("lm", knn=knnreg)`, the number of nearest neighbors for knn can be set with `list(NULL, list(k = 7))`.

`pool.size`
: specifies the number of candidate elements to be sampled from the unlabeled set `U`. The best candidate elements from the pool are labeled and added to the training set. The `gr` parameter controls how many of the best candidates are used to augment the training set at each iteration. This parameter has big influence in computational time since in each iteration, `pool.size * length(regressors)` models are trained and evaluated in order to find the best candidate data points.

`gr`
: an integer specifying the *growth rate*, i.e., how many of the best elements from the pool are added to the training set for each base model at each iteration.

`maxits`
: an integer that specifies the maximum number of iterations. The training phase will terminate either when `maxits` is reached or when `U` becomes empty.

`testdata`
: a data frame containing the test set to be evaluated within each iteration. If `verbose = TRUE` and `plotmetrics = TRUE` the predictive performance of the model on the test set will be printed/plotted for each iteration.

`shuffle`
: a boolean specifying whether or not to shuffle the data frames rows before training the models. Defaults to `TRUE`. Some models like neural networks are sensitive to row ordering. Often, you may want to shuffle before training.

`verbose`
: a boolean specifying whether or not to print diagnostic information to the console within each iteration. If `testdata` is provided, the information includes performance on the test set such as RMSE and improvement percent with respect to the initial model before data from `U` was used. Default is `TRUE`.

`plotmetrics`
: a boolean that specifies if performance metrics should be plotted for each iteration when `testdata` is provided. Default is `FALSE`.

`U.y`
: an optional numeric vector with the true values fo the response variable for the unlabeled set `U`. If this parameter is `!= NULL` then, the true values will be used to determine the best candidates to augment the training set and the true values will be kept when adding them to the training set. *This parameter should be used with caution* and is intended to be used to generate an upper bound model for comparison purposes only. This is to simulate the case when the model can label the unlabeled data points used to augment the training set with 100% accuracy.

Details

The co-training by committee implementation is based on Hady et al. (2009). It consists of a set of *n* base models (the committee), each, initially trained with independent bootstrap samples from the labeled training set `L`. The Out-of-Bag (OOB) elements are used for validation. The training set for

each base model *b* is augmented by selecting the most relevant elements from the unlabeled data set U. To determine the most relevant elements for each base model *b*, the other models (excluding *b*) label a set of data pool.size points sampled from U by taking the average of their predictions. For each newly labeled data point, the base model *b* is trained with its current labeled training data plus the new data point and the error on its OOB validation data is computed. The top gr points that reduce the error the most are kept and used to augment the labeled training set of *b* and removed from U.

When the regressors list contains a single model, *self-learning* is performed. In this case, the base model labels its own data points as opposed to co-training by committee in which the data points for a given model are labeled by the other models.

In the original paper, Hady et al. (2009) use the same type of regressor for the base models but with different parameters to introduce diversity. The ssr function allows the user to specify any type of regressors as the base models. The regressors can be models from the 'caret' package, other packages, or custom functions. Models from other packages or custom functions need to comply with certain structure. First, the model's function used for training must have a formula as its first parameter and a parameter named data that accepts a data frame as the training set. Secondly, the predict() function must have the trained model as its first parameter. Most of the models from other libraries follow this pattern. If they do not follow this pattern, you can still use them by writing a wrapper function. To see examples of all those cases, please check the vignettes.

Value

A list object of class "ssr" containing:

models A list of the final trained models in the last iteration.

formula The formula provided by the user in theFormula.

regressors The list of initial regressors set by the user with formatted names.

regressors.names The names of the regressors names(regressors).

regressors.params The initial list of parameters provided by the user.

pool.size The initial pool.size specified by the user.

gr The initial gr specified by the user.

testdata A boolean indicating if test data was provided by the user: !is.null(testdata).

U.y A boolean indicating if U.y was provided by the user: !is.null(U.y).

numits The total number of iterations performed by the algorithm.

shuffle The initial shuffle value specified by the user.

valuesRMSE A numeric vector with the Root Mean Squared error on the testdata for each iteration. The length is the number of iterations + 1. The first position valuesRMSE[1] stores the initial RMSE before using any data from U.

valuesRMSE.all A numeric matrix with *n* rows and *m* columns. Stores Root Mean Squared Errors of the individual regression models. The number of rows is equal to the number of iterations + 1 and the number of columns is equal to the number of regressors. A column represents a regressor in the same order as they were provided in regressors. Each row stores the RMSE for each iteration and for each regression model. The first row stores the initial RMSE before using any data from U.

valuesMAE Stores Mean Absolute Error information. Equivalent to **valuesRMSE**.

valuesMAE.all Stores Mean Absolute Errors of the individual regression models. Equivalent to **valuesRMSE.all**

valuesCOR Stores Pearson Correlation information. Equivalent to **valuesRMSE**.

valuesCOR.all Stores the Pearson Correlation of the individual regression models. Equivalent to **valuesRMSE.all**

References

Hady, M. F. A., Schwenker, F., & Palm, G. (2009). Semi-supervised Learning for Regression with Co-training by Committee. In International Conference on Artificial Neural Networks (pp. 121-130). Springer, Berlin, Heidelberg.

Examples

```
dataset <- friedman1 # Load friedman1 dataset.

set.seed(1234)

# Split the dataset into 70% for training and 30% for testing.
split1 <- split_train_test(dataset, pctTrain = 70)

# Choose 5% of the train set as the labeled set L and the remaining will be the unlabeled set U.
split2 <- split_train_test(split1$trainset, pctTrain = 5)

L <- split2$trainset

U <- split2$testset[, -11] # Remove the labels.

testset <- split1$testset

# Define list of regressors. Here, only one regressor (KNN). This trains a self-learning model.
# For co-training by committee, add more regressors to the list. See the vignettes for examples.
regressors <- list(knn = caret::knnreg)

# Fit the model.
model <- ssr("Ytrue ~ .", L, U, regressors = regressors, testdata = testset, maxits = 10)

# Plot RMSE.
plot(model)

# Get the predictions on the testset.
predictions <- predict(model, testset)

# Calculate RMSE on the test set.
sqrt(mean((predictions - testset$Ytrue)^2))
```

Index