Package ‘iForecast’

January 8, 2025

Type Package

Title Machine Learning Time Series Forecasting

Version 1.1.0

Date 2025-01-08

Author Ho Tsung-wu [aut, cre]

Maintainer Ho Tsung-wu <tsungwu@ntnu.edu.tw>

Description Compute static, onestep and multistep time series forecasts for machine learning models.

License GPL (>= 2)

LazyData TRUE

LazyLoad yes

Depends R (>= 3.5)

Imports caret,zoo

Suggests forecast, h2o, kernlab, lubridate, timeSeries, timeDate, xts

NeedsCompilation no

Repository CRAN

Date/Publication 2025-01-08 08:10:02 UTC

Contents

`Accuracy` *Accuracy measures for a forecast model*

Description

Returns range of summary measures of the forecast accuracy. Except MAAPE, all measures are defined and discussed in Hyndman and Koehler (2006).

Usage

```
Accuracy(f,x)
```

Arguments

`f`	A time series forecasting object generated by `iForecast`.
`x`	Actual values of the same length as the time series object of `f`.

Details

The measures calculated are:

- RMSE: Root Mean Squared Error
- MAE: Mean Absolute Error
- MAPE: Mean Absolute Percentage Error
- MAAPE: Mean Absolute Arctan Percentage Error
- ACF1: Autocorrelation of errors at lag 1.

Except MAAPE, by default, see Hyndman and Koehler (2006) and Hyndman and Athanasopoulos (2014, Section 2.5) for further details. For MAAPE, please see Kim and Kim (2016).

Value

Matrix giving forecast accuracy measures.

Author(s)

Ho Tsung-wu <tsungwu@ntnu.edu.tw>, College of Management, National Taiwan Normal University.

References

Hyndman, R.J. and Koehler, A.B. (2006) "Another look at measures of forecast accuracy". *International Journal of Forecasting*, **22**(4), 679-688.
Hyndman, R.J. and Athanasopoulos, G. (2018) "Forecasting: principles and practice", 2nd ed., OTexts, Melbourne, Australia. Section 3.4 "Evaluating forecast accuracy".<https://otexts.com/fpp2/accuracy.html>
Kim Sungil and Heeyoung Kim (2016) "A new metric of absolute percentage error for intermittent demand forecasts", *International Journal of Forecasting*,**32**(3),669-679. <https://doi.org/10.1016/j.ijforecast.2015.12.003>.

Examples

```
tmp0=timeSeries::as.timeSeries(ts(rnorm(800),start=c(1960,1),freq=12))
fit1 <- timeSeries::as.timeSeries(forecast::rwf(tmp0[1:700,1],h=100)$mean)
Accuracy(f=fit1,x=tmp0[701:800,1])
```

---

data-sets *Economic and Financial Data Sets*

Description

ES_15m is 15-min realized absolute variance of e-mini S&P 500. macrodata contains monthly US unemployment(unrate), ES_Daily is daily realized absolute variance of e-mini S&P 500. macrodata contains monthly US unemployment(unrate) and and year-to-year changes in three regional business cycle indices (OECD, NAFTA, and G7). bc contains monthly business cycle data, bc is binary indicator(0=recession, 1=boom) of Taiwan's business cycle phases, IPI_TWN is industrial production index of Taiwan, LD_OECD, LD_G7, and LD_NAFTA are leading indicators of OECD, G7 and NAFTA regions; all four are monthly rate of changes.

Usage

```
data(ES_15m)
data(macrodata)
data(ES_Daily)
data(bc)
```

Value

an object of class "zoo".

---

iForecast *Extract predictions and class probabilities from train objects*

Description

It generates both the static and recursive time series plots of machine learning prediction object generated by ttsCaret, ttsAutoML and ttsLSTM.

Usage

```
iForecast(Model,newdata,Type,a.head)
```

Arguments

- `Model` Object of trained model.
- `newdata` The dataset for pediction, the column names must be the same as the trained data.Not required if type="dynamic".
- `Type` If Type="static", it computes the (static) forecasting values of insample model fit. If Type="dynamic", it recursively computes the forecasting values ahead.
- `a.head` For Type="dynamic", it is the number of forecasting periods ahead. Not required if type="static".

Details

This function generates forecasts of tts.caret,and tts.autoML.

Value

- `prediction` The forecasted time series target variable. For binary case, it returns both porbabilities and class.

Author(s)

Ho Tsung-wu <tsungwu@ntnu.edu.tw>, College of Management, National Taiwan Normal University.

Examples

```
# Cross-validation takes time, example below is commented.
## Machine Learning by library(caret)
#Case 1. Low frequency, regression type
data("macrodata")
dep <- macrodata[569:669,"unrate",drop=FALSE]
ind <- macrodata[569:669,-1,drop=FALSE]
train.end <- "2018-12-01"# Choosing the end dating of train

models <- c("svm","rf","rpart","gbm","nb")[1]
type <- c("none","trend","season","both")[1]
# output <- tts.caret(y=dep, x=ind, arOrder=c(1), xregOrder=c(1),
# method="smv", tuneLength =1,
# train.end, type=type,resampling="cv",preProcess = "center")

# testData1 <- window(output$dataused,start="2019-01-01",end=end(output$dataused))
#P1=iForecast(Model=output,Type="static",newdata=testData1)
#P2=iForecast(Model=output,Type="dynamic",a.head=7)

#tail(cbind(testData1[,1],P1))
#tail(cbind(testData1[,1],P2))

#Case 2. Low frequency, binary type
data(bc) #binary dependent variable, business cycle phases
dep=bc[,1,drop=FALSE]
ind=bc[,-1]
```

```
train.end=as.character(rownames(dep))[as.integer(nrow(dep)*0.8)]
test.start=as.character(rownames(dep))[as.integer(nrow(dep)*0.8)+1]

#output = tts.caret(y=dep, x=ind, arOrder=c(1), xregOrder=c(1), method="nb",
#                   tuneLength =10, train.end, type=type)

#testData1=window(output$dataused,start=test.start,end=end(output$dataused))

#head(output$dataused)
#P1=iForecast(Model=output,Type="static",newdata=testData1)
#P2=iForecast(Model=output,Type="dynamic",a.head=7)

#tail(cbind(testData1[,1],P1),10)
#tail(cbind(testData1[,1],P2),10)
```

`iForecast-ttsAutoML` *Defunct functions in package 'iForecast'*

Description

These functions are defunct and no longer available.

Details

Defunct function is: `ttsAutoML` New function is: `tts.autoML`

`iForecast-ttsCaret` *Defunct functions in package 'iForecast'*

Description

These functions are defunct and no longer available.

Details

Defunct function is: `ttsCaret` New function is: `tts.caret`

`iForecast-ttsLSTM` *Defunct functions in package 'iForecast'*

Description

These functions are defunct and no longer available.

Details

Defunct functions are: `ttsLSTM`

`rollingWindows` *Rolling timeframe for time series anaysis*

Description

It extracts time stamp from a timeSeries object and separates the time into in-sample training and out-of-sample validation ranges.

Usage

```
rollingWindows(x,estimation="18m",by = "1m")
```

Arguments

`x`	The time series object with `timeSeries`, `xts`, or `zoo` format of "
`estimation`	The range of insample estimation period, the default is 18 months(18m), where the k-fold cross-section is performed. Quarter, week and day are also supported (see example).
`by`	The range of out-of-sample validation/testing period, the default is 6 months(6m).Quarter, week and day are also supported (see example).

Details

This function is similar to the backtesting framework in portfolio analysis. Rolling windows fixes the origin and the training sample grows over time, moving windows can be achieved by placing window() on dependent variable at each iteration.

Value

`window`	The time labels of from and to

.

Author(s)

Ho Tsung-wu <tsungwu@ntnu.edu.tw>, College of Management, National Taiwan Normal University.

Examples

```
data(macrodata)
y=macrodata[,1,drop=FALSE]
timeframe=rollingWindows(y,estimation="300m",by="6m")
#estimation="300m", because macrodata is monthly
FROM=timeframe$from
TO=timeframe$to

data(ES_Daily)
y=ES_Daily[,1,drop=FALSE]
```

```
timeframe=rollingWindows(y,estimation  ="60w",by="1w")
#60 weeks(300+ days) as estimation window and move by 1 week(5+ days).

FROM=timeframe$from
TO=timeframe$to

y=ES_Daily[,1,drop=FALSE]
timeframe=rollingWindows(y,estimation  ="250d",by="10d")
#250-day as estimation window and move by 10 days.


# simulated quarterly data
tmp0=ts(rnorm(800),start=c(1900,1),freq=4)
tmp1=timeSeries::as.timeSeries(tmp0)
tmp2=zoo::as.zoo(tmp0)
tmp3=xts::as.xts(tmp0)
timeframe=rollingWindows(x=tmp3,estimation ="100q",by="12q")
FROM=timeframe$from
TO=timeframe$to
```

`tts.autoML` *Train time series by automatic machine learning of* `h2o` *provided by H2O.ai*

Description

It generates both the static and recursive time series plots of H2O.ai object generated by package `h2o` provided by H2O.ai.

Usage

```
tts.autoML(y,x=NULL,train.end,arOrder=2,xregOrder=0,type,max_models = 20,
                        sort_metric="AUTO",stopping_metric = "AUTO")
```

Arguments

`y`	The time series object of the target variable, for example, `timeSeries`,`xts`, or `zoo`. Numerically,y must be real numbers for regression or integers for classification. Date format must be "
`x`	The time series matrix of input variables, timestamp is the same as y, maybe null.
`train.end`	The end date of training data, must be specificed. The default dates of train.start and test.end are the start and the end of input data; and the test.start is the 1-period next of train.end.
`arOrder`	The autoregressive order of the target variable, which may be sequentially specifed like arOrder=1:5; or discontinuous lags like arOrder=c(1,3,5); zero is not allowed.

`xregOrder`	The distributed lag structure of the input variables, which may be sequentially specifed like xregOrder=1:5; or discontinuous lags like xregOrder=c(0,3,5); zero is allowed since contemporaneous correlation is allowed.
`type`	The time dummies variables. We have four selection: 'none'=no other variables, 'trend'=inclusion of time dummy, 'season'=inclusion of seasonal dummies, 'both'=inclusion of both trend and season. No default.
`max_models`	Number of AutoML base models, default to 20.
`sort_metric`	Specifies the metric used to sort the Leaderboard by at the end of an AutoML run. Defaults to "AUTO", where 'AUC' (area under the ROC curve) for binary classification, 'mean_per_class_error' for multinomial classification, and 'deviance' for regression. Available options include:'MSE','RMSE','MAE','RMSLE','AUCPR' (area under the Precision-Recall curve)
`stopping_metric`	Specify the metric to use for early stopping. Defaults to "AUTO",where 'logloss' for classification and 'deviance' for regression. Besides, options are: 'MSE','RMSE','MAE','RMSLE','A

Details

This function calls the h2o.automl function from package `h2o` to execute automatic machine learning estimation. When execution finished, it computes two types of time series forecasts: static and recursive. The procedure of h2o.automl automatically generates a lot of time features.

Value

`output`	Output object generated by h2o.automl function of `h2o`.
`modelsUsed`	AutoML Leaderboard object, which is a table returns the argument of 'max_models'.
`arOrder`	The autoregressive order of the target variable used.
`dataused`	The data used by arOrder, xregOrder
`data`	The complete data structure
`TD`	Time dummies used, inherited from 'type' in tts.autoML
`train.end`	The same as the argument in tts.autoML

Author(s)

Ho Tsung-wu <tsungwu@ntnu.edu.tw>, College of Management, National Taiwan Normal University.

Examples

```
# Computation takes time, example below is commented.
data("macrodata")
dep<-macrodata[,"unrate",drop=FALSE]
ind<-macrodata[,-1,drop=FALSE]

# Choosing the dates of training and testing data
```

```
train.end<-"2008-12-01"

#autoML of H2O.ai

# autoML <- tts.autoML(y=dep, x=ind, train.end,arOrder=c(2,4),
# xregOrder=c(0,1,3),type="both")
# print(autoML$modelsUsed,n=22) #View the AutoML Leaderboard

#testData2 <- window(autoML$dataused,start="2009-01-01",end=end(autoML$dataused))
#P1<-iForecast(Model=autoML,Type="static",newdata=testData2)
#P2<-iForecast(Model=autoML,Type="dynamic",a.head=nrow(testData2))

#tail(cbind(testData2[,1],P1))
#tail(cbind(testData2[,1],P2))
```

`tts.caret` *Train time series by* `caret` *and produce two types of time series forecasts: static and dynamic*

Description

It generates both the static and dynamic time series plots of machine learning prediction object generated by package `caret`.

Usage

```
tts.caret(
  y,
  x=NULL,
  method,
  train.end,
  arOrder=2,
  xregOrder=0,
  type,
  tuneLength =10,
  preProcess = NULL,
  resampling="boot",
  Number=NULL,
  Repeat=NULL)
```

Arguments

`y`	The time series object of the target variable, for example, `timeSeries`,`xts`, or `zoo`. y can be either binary or continuous. Date format must be "
`x`	The time series matrix of input variables, timestamp is the same as y, maybe null.

`method` The train_model_list of `caret`. While using this, make sure that the method allows regression. Methods in c("svm","rf","rpart","gamboost","BstLm","bstSm","blackboost") are feasible.

`train.end` The end date of training data, must be specificed.The default dates of train.start and test.end are the start and the end of input data; and the test.start is the 1-period next of train.end.

`arOrder` The autoregressive order of the target variable, which may be sequentially specifed like arOrder=1:5; or discontinuous lags like arOrder=c(1,3,5); zero is not allowed.

`xregOrder` The distributed lag structure of the input variables, which may be sequentially specifed like xregOrder=0:5; or discontinuous lags like xregOrder=c(0,3,5); zero is allowed since contemporaneous correlation is allowed.

`type` The time dummies variables. We have four selection:
"none"=no other variables,
"trend"=inclusion of time dummy,
"season"=inclusion of seasonal dummies,
"both"=inclusion of both trend and season. No default.

`tuneLength` The same as the length specified in train function of package `caret`.

`preProcess` Whether to pre-process the data, current possibilities are "BoxCox", "YeoJohnson", "expoTrans", "center", "scale", "range", "knnImpute", "bagImpute", "medianImpute", "pca", "ica" and "spatialSign".The default is no pre-processing.

`resampling` The method for resampling, as trainControl function list in package `caret`. The default is "boot" for bootstrapping with 25 replications. Current choices are c("cv","boot","repeatedcv","LOOCV") where "cv" is K-fold CV with a default K=10 or specified by the "Number" below, "LOOCV" denotes the leave-one-out CV

`Number` The number of K for K-Fold CV, default (NULL) is 10; for "boot" option, the default number of replications is 25

`Repeat` The number for the repeatition for "repeatedcv".

Details

This function calls the train function of package `caret` to execute estimation. When execution finished, we compute two types of time series forecasts: static and recursive.

Value

`output` Output object generated by train function of `caret`.

`arOrder` The autoregressive order of the target variable used.

`training.Pred` All tuned prediction values of training data, using besTunes to extract the best prediction.

`dataused` The data used by arOrder, xregOrder, and type.

`data` The complete data structure

`TD` Time dummies used, inherited from type

`train.end` The same as argument in tts.caret

Author(s)

Ho Tsung-wu <tsungwu@ntnu.edu.tw>, College of Management, National Taiwan Normal University.

Examples

```
# Cross-validation takes time, example below is commented.
## Machine Learning by library(caret)
library(zoo)
#Case 1. Low frequency
data("macrodata")
dep <- macrodata[569:669,"unrate",drop=FALSE]
ind <- macrodata[569:669,-1,drop=FALSE]
train.end <- "2018-12-01"# Choosing the end dating of train

models <- c("glm","knn","nnet","rpart","rf","svm","enet","gbm","lasso","bridge","nb")[2]
type <- c("none","trend","season","both")[1]
output <- tts.caret(y=dep,x=NULL, arOrder=c(1), xregOrder=c(1),
 method=models, tuneLength =1, train.end, type=type,
 resampling=c("boot","cv","repeatedcv")[1],preProcess = "center")

testData1 <- window(output$dataused,start="2019-01-01",end=end(output$dataused))
P1 <- iForecast(Model=output,Type="static",newdata=testData1)
P2 <- iForecast(Model=output,Type="dynamic",a.head=nrow(testData1))

tail(cbind(testData1[,1],P1,P2))

#Case 2. High frequency
#head(ES_15m)
#head(ES_Daily)
#dep <- ES_15m #SP500 15-minute realized absolute variance
#ind <- NULL
#train.end <- as.character(rownames(dep))[as.integer(nrow(dep)*0.9)]

#models<-c("svm","rf","rpart","gamboost","BstLm","bstSm","blackboost")[1]
#type<-c("none","trend","season","both")[1]
# output <- tts.caret(y=dep, x=ind, arOrder=c(3,5), xregOrder=c(0,2,4),
# method=models, tuneLength =10, train.end, type=type,
# resampling=c("boot","cv","repeatedcv")[2],preProcess = "center")
#testData1<-window(output$dataused,start="2009-01-01",end=end(output$dataused))
#P1<-iForecast(Model=output,Type="static",newdata=testData1)
#P2<-iForecast(Model=output,Type="dynamic",a.head=nrow(testData1))
```

Index