Package ‘predtoolsTS’

Type Package

Title Time Series Prediction Tools

Version 0.1.1

Date 2018-04-26

Author Alberto Vico Moreno [aut, cre],
Antonio Jesus Rivera Rivas [aut, ths],
Maria Dolores Perez Godoy [aut, ths]

Maintainer Alberto Vico Moreno <avm00016@red.ujaen.es>

Description Makes the time series prediction easier by automatizing this process using four main functions: prep(), modl(), pred() and postp(). Features different preprocessing methods to homogenize variance and to remove trend and seasonality. Also has the potential to bring together different predictive models to make comparatives. Features ARIMA and Data Mining Regression models (using caret).

License GPL (>= 2)

Encoding UTF-8

LazyData true

URL https://github.com/avm00016/predtoolsTS

RoxygenNote 6.0.1

Imports caret, forecast, graphics, methods, Metrics, stats, TSPred, tseries, utils

NeedsCompilation no

Repository CRAN

Date/Publication 2018-04-29 13:54:50 UTC

R topics documented:

  modl ................................................................. 2
  modl.arima .................................................................. 4
  modl.dataMining .......................................................... 5
  modl.trControl ........................................................... 6
modl

Building predictive models

Description

This function gives us the tools to build predictive models for time series.

Usage

```
modl(tserie, method = "arima", algorithm = NULL, formula = NULL,
     initialWindow = NULL, horizon = NULL, fixedWindow = NULL)
```
**Arguments**

- **tserie**: A ts or prep object.
- **method**: A string. Current methods available are "arima" and "dataMining". Method "arima" is set as default.
- **algorithm**: A string. In case method is "dataMining", pick the algorithm you want to use. There is a complete list of available algorithms here (only regression type allowed): [http://topepo.github.io/caret/train-models-by-tag.html](http://topepo.github.io/caret/train-models-by-tag.html).
- **formula**: An integer vector. Contains the indexes from the time series which will indicate how to extract the features. The last value will be the class index. Default value: c(1:16)
- **initialWindow**: An integer. The initial number of consecutive values in each training set sample. Default value: 30.
- **horizon**: An integer. The number of consecutive values in test set sample. Default value: 15.
- **fixedWindow**: A logical: if FALSE, the training set always start at the first sample and the training set size will vary over data splits. Default value: TRUE.

**Details**

Returns an object **modl** which stores all the information related to the final chosen model (errors, parameters, model).

Currently this function covers two different methods: the widely known ARIMA and the "not so used for prediction" data mining. For the data mining we make use of the caret package. The caret package offers plenty of data mining algorithms. For the data splitting here we use a rolling forecasting origin technique, which works better on time series.

**Value**

A list is returned of class **modl** containing:

- **tserie**: Original time serie.
- **tserieDF**: Time serie converted to data frame.
- **method**: Method used to build the model.
- **algorithm**: If method is data mining, indicates which algorithm was used.
- **horizon**: Horizon for the splitting.
- **model**: Model result from caret. It is a list, result of the caret::train function.
- **errors**: Contains three different metrics to evaluate the model.

**Author(s)**

Alberto Vico Moreno

**References**


modl.arima

See Also
prep modl.arima, modl.tsToDataFrame, modl.trControl, modl.dataMining

Examples

```r
p <- prep(AirPassengers)
modl(p, method='arima')
modl(p, method='dataMining', algorithm='rpart')
```

modl.arima  
*Automatic ARIMA model*

Description
Assuming "tserie" is stationary, returns the best arima model

Usage

```r
modl.arima(tserie)
```

Arguments

```r
tserie  
A ts object.
```

Value

ARIMA model.

Author(s)
Alberto Vico Moreno

Examples

```r
modl.arima(AirPassengers)
```
modl.dataMining

Train the data

Description

Train the time serie (as data frame) to build the model.

Usage

modl.dataMining(form, tserieDF, algorithm, timeControl, metric = "RMSE", maximize = FALSE)

Arguments

form
A formula of the form y ~ x1 + x2 + ...

tserieDF
Data frame.

algorithm

timeControl
trainControl object.

metric
A string. Specifies what summary metric will be used to select the optimal model. Possible values in caret are "RMSE" and "R squared". "RMSE" set as default. If you used a custom summaryFunction (see ?trainControl) your metrics will prevail over default.

maximize
A logical. Should the metric be maximized or minimized? Default is FALSE, since that is what makes sense for time series.

Value

train object

Author(s)
Alberto Vico Moreno

Examples

modl.dataMining(form=Class ~ .,
tserieDF=modl.tsToDataFrame(AirPassengers, formula=c(1:20)),
algorithm='rpart',
timeControl=modl.trControl(initialWindow=30, horizon=15, fixedWindow=TRUE))
modl.trControl  Control the splitting to train the data

Description

Creates the needed caret::trainControl object to control the training splitting.

Usage

modl.trControl(initialWindow, horizon, fixedWindow, givenSummary = FALSE)

Arguments

initialWindow  An integer. The initial number of consecutive values in each training set sample. Default value: 30.
horizon       An integer. The number of consecutive values in test set sample. Default value: 15.
fixedWindow   A logical: if FALSE, the training set always start at the first sample and the training set size will vary over data splits. Default value: TRUE.
givenSummary  A logical. Indicates if it should be used the customized summaryFunction(?trainControl for more info) modl.sumFunction or not. Default is FALSE; this will use default caret metrics.

Details

We always split using method "timeslice", which is the better for time series. More information on how this works on http://topepo.github.io/caret/data-splitting.html#data-splitting-for-time-series.

Value

trainControl object

Author(s)

Alberto Vico Moreno

Examples

modl.trControl(initialWindow=30, horizon=15, fixedWindow=TRUE, givenSummary=TRUE)
modl.tsToDataFrame

Ts to data frame transformation

Description

Transform a ts object into a data frame using the given formula.

Usage

modl.tsToDataFrame(tserie, formula = NULL)

Arguments

- `tserie` A ts object.
- `formula` An integer vector. Contains the indexes from the tserie which will indicate how to extract the features. The last value will be the class index. Default value: c(1:16). Has to be length 6 minimum.

Value

the time serie as data frame

Author(s)

Alberto Vico Moreno

Examples

modl.tsToDataFrame(AirPassengers, formula=c(1,3,4,5,6,7))
modl.tsToDataFrame(AirPassengers, formula=c(1:20))

plot.pred

Generic function

Description

Plots object prep

Usage

## S3 method for class 'pred'
plot(x, ylab = "Values", main = "Predictions", ...)
plot.prep

Arguments

  x  pred object
  ylab  ylab
  main  main
  ...  ignored

Examples

plot(pred(modl(prep(AirPassengers)))))

plot.prep  Generic function

Description

Plots object prep

Usage

  ## S3 method for class 'prep'
  plot(x, ylab = "Preprocessed time serie", xlab = "", ...)

Arguments

  x  prep object
  ylab  ylab
  xlab  xlab
  ...  ignored

Examples

plot(prep(AirPassengers), ylab="Stationary AirPassengers")
**postp**  

*Post-processing of pre-processed data*

---

**Description**

Using the prep data we undo the changes on a pred object.

**Usage**

```
postp(prd, pre)
```

**Arguments**

- `prd`: A pred object.
- `pre`: A prep object.

**Value**

A pred object with reverted transformations.

**Author(s)**

Alberto Vico Moreno

**See Also**

`predprep, postp.homogenize.log, postp.homogenize.boxcox, postp.detrend.differencing, postp.detrend.sfsm, postp.deseason.differencing`

**Examples**

```r
preprocess <- prep(AirPassengers)
prediction <- pred(modl(preprocess), n.ahead=30)
postp.prediction <- postp(prediction, preprocess)
```

---

**postp.deseason.differencing**

*Undo deseason(differencing)*

---

**Description**

Uses inverse seasonal differences to reverse the changes

**Usage**

```
postp.deseason.differencing(tserie, nsd, firstseasons, frequency)
```
Arguments

- `tserie` A ts object.
- `nsd` Number of seasonal differences.
- `firstseasons` Values lost on the original differences.
- `frequency` Frequency of the original time serie.

Value

A ts object.

Author(s)

Alberto Vico Moreno

Examples

```r
p <- prep.deseason.differencing(AirPassengers)
postp.deseason.differencing(p$tserie, p$nsd, p$firstseasons, frequency(AirPassengers))
```

Description

Uses inverse differences to revert the changes.

Usage

```r
postp.detrend.differencing(tserie, nd, firstvalues)
```

Arguments

- `tserie` A ts object.
- `nd` Number of differences.
- `firstvalues` Values lost on the original differences.

Value

A ts object.

Author(s)

Alberto Vico Moreno

Examples

```r
p <- prep.detrend.differencing(AirPassengers)
postp.detrend.differencing(p$tserie, p$nd, p$firstvalues)
```
postp.detrend.sfsm  

**Description**

Undo detrend (subtracting full-means method)

**Usage**

`postp.detrend.sfsm(tserie, means, start, frequency)`

**Arguments**

- `tserie`  
  A `ts` object.
- `means`  
  A numeric vector.
- `start`  
  Start of original time serie
- `frequency`  
  Frequency of the original time serie

**Value**

A `ts` object.

**Author(s)**

Alberto Vico Moreno

**Examples**

```r
p <- prep.detrend.sfsm(AirPassengers)
postp.detrend.sfsm(p$tserie, p$means, start(AirPassengers), frequency(AirPassengers))
```

postp.homogenize.boxcox

**Description**

Undo Box-Cox transformation

**Usage**

`postp.homogenize.boxcox(tserie, lambda)`


**Arguments**

- `tserie` A ts object.
- `lambda` A numeric.

**Value**

A ts object.

**Author(s)**

Alberto Vico Moreno

**Examples**

```r
p <- prep.homogenize.boxcox(AirPassengers)
postp.homogenize.boxcox(p$tserie, p$lambda)
```

---

**Description**

Uses exponent to reverse the logarithm

**Usage**

```r
postp.homogenize.log(tserie)
```

**Arguments**

- `tserie` A ts object.

**Value**

A ts object.

**Author(s)**

Alberto Vico Moreno

**Examples**

```r
postp.homogenize.log(prep.homogenize.log(AirPassengers))
```
Description

Performs predictions over a trained model.

Usage

pred(model = NULL, n.ahead = 20, tserie = NULL, predictions = NULL)

Arguments

model A `modl` object. Contains the trained model we want to predict with.
n.ahead Number of values to predict ahead of the end of the original time serie. Default value is 20. Must ve lower than 100.
tserie A `ts` object.
predictions A `ts` object.

Details

Predicts future values over a "modl" object which can be ARIMA or data mining, and returns the predictions. Data mining predictions start right after the last value contained in the training data, so they overlap with the end of the original.

The object contains only two time series: the original one and the predictions. You can just set these series aswell.

Value

A list is returned of class `pred` containing:

tserie Original time serie.
predictions Time serie with the predictions.

Author(s)

Alberto Vico Moreno

See Also

`modl`, `pred.arima`, `pred.dataMining`, `pred.compareModels`

Examples

```r
prediction <- pred(model=modl(prep(AirPassengers)),n.ahead=25)
pred(tserie=prediction$tserie, predictions=prediction$predictions)
```
pred.arima  

**Predicts for ARIMA**

### Description

Performs predictions over an ARIMA model using the `stats::predict` function.

### Usage

```r
pred.arima(model, n.ahead)
```

### Arguments

- **model**: An ARIMA model.
- **n.ahead**: Number of values to predict.

### Value

A `ts` object containing the predictions.

### Author(s)

Alberto Vico Moreno

### Examples

```r
pred.arima(forecast::auto.arima(prep(AirPassengers)$tserie), n.ahead = 30)
```

---

pred.compareModels  

**Compare different predictions**

### Description

Plots the original time serie along with 2-5 predictive models.

### Usage

```r
pred.compareModels(originalTS, p_1, p_2, p_3 = NULL, p_4 = NULL, p_5 = NULL, legendNames = NULL, colors = NULL, legend = TRUE, legendPosition = NULL, yAxis = "Values", title = "Predictions")
```
pred.compareModels

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>originalTS</td>
<td>A ts object</td>
</tr>
<tr>
<td>p_1</td>
<td>A ts object</td>
</tr>
<tr>
<td>p_2</td>
<td>A ts object</td>
</tr>
<tr>
<td>p_3</td>
<td>A ts object. Default is NULL.</td>
</tr>
<tr>
<td>p_4</td>
<td>A ts object. Default is NULL.</td>
</tr>
<tr>
<td>p_5</td>
<td>A ts object. Default is NULL.</td>
</tr>
<tr>
<td>legendNames</td>
<td>String vector with the names for the legend. Has to be same length as number</td>
</tr>
<tr>
<td></td>
<td>of time series we are plotting(including the original one). Default is NULL.</td>
</tr>
<tr>
<td>colors</td>
<td>Vector with the colors. Has to be same length as number of time series we are</td>
</tr>
<tr>
<td></td>
<td>plotting(including the original one). Default is NULL.</td>
</tr>
<tr>
<td>legend</td>
<td>A logical. Do we want a legend? Default is TRUE.</td>
</tr>
<tr>
<td>legendPosition</td>
<td>A string with the position of the legend (bottomright, topright, ...). Defa</td>
</tr>
<tr>
<td></td>
<td>ult is NULL.</td>
</tr>
<tr>
<td>yAxis</td>
<td>A string. Name for the y axis. &quot;Values&quot; as default.</td>
</tr>
<tr>
<td>title</td>
<td>A string. Title for the plot. &quot;Predictions&quot;.</td>
</tr>
</tbody>
</table>

Details

This function aims to ease the comparation between different predictive models by plotting them into the same graphic.

Author(s)

Alberto Vico Moreno

Examples

data(AirPassengers)
#pre-processing
p <- prep(AirPassengers)
#modelling
arima.modl <- modl(p)
cart.modl <- modl(p,method='dataMining',algorithm='rpart')
#predicting
arima.pred <- pred(arima.modl,n.ahead=30)
cart.pred <- pred(cart.modl,n.ahead=45)
#post-processing
arima.pred <- postp(arima.pred,p)
cart.pred <- postp(cart.pred,p)
#visual comparison
pred.compareModels(AirPassengers,arima.pred$predictions,cart.pred$predictions
,legendNames=c('AirPassengers','ARIMA','CART'),yAxis='Passengers',legendPosition = 'topleft')
pred.dataMining  

Predicts for data mining methods

Description

Performs predictions over a data mining model using the caret::predict.train function.

Usage

pred.dataMining(model, n.ahead)

Arguments

model  A modl object.

n.ahead  Number of values to predict.

Value

A ts object containing the predictions.

Author(s)

Alberto Vico Moreno

Examples

m <- modl(prep(AirPassengers),method='dataMining',algorithm='rpart')
pred.dataMining(m,n.ahead=15)

prep  

Automatic pre-preprocessing

Description

This function performs pre-processing on a time series object(ts) to treat heterocedasticity, trend and seasonality in order to make the serie stationary.

Usage

prep(tserie, homogenize.method = "log", detrend.method = "differencing", nd = NULL, deseason.method = "differencing", nsd = NULL, detrend.first = TRUE)
Arguments

- **tserie** A ts object.
- **homogenize.method** A string. Current methods available are "log" and "boxcox". Method "log" is set as default. If you don’t want to perform this transformation, set method as "none".
- **detrend.method** A string. Current methods available are "differencing" and "sfsm". Method "differencing" is set as default. If you don’t want to perform this transformation, set method as "none".
- **nd** A number. Number of differences you want to apply to the "differencing" detrending method. As default its value is NULL, which means nd will be calculated internally.
- **deseason.method** A string. Current methods available are "differencing". Method "differencing" is set as default. If you don’t want to perform this transformation, set method as "none".
- **nsd** A number. Number of seasonal differences you want to apply to the "differencing" deseasoning method. As default its value is NULL, which means nsd will be calculated internally.
- **detrend.first** A boolean. TRUE if detrending method is applied first, then deseasoning. FALSE if deseasoning method is applied first. Default is TRUE.

Details

Returns an object prep which stores all data needed to undo the changes later on.

This function provides an automatic way of pre-processing based on unit root tests, but this is not the perfect way to do it. You should always check manually if the given time serie is actually stationary, and modify the parameters according to your thoughts.

Value

A list is returned of class prep containing:

- **tserie** Processed ts object.
- **homogenize.method** Method used for homogenizing.
- **detrend.method** Method used for detrending.
- **nd** Number of differences used on detrending through differencing.
- **firstvalues** First nd values of the original series.
- **deseason.method** Method used for deseasoning.
- **nsd** Number of seasonal differences used on deseasoning through differencing.
- **firstseasons** First nsd seasons of the original series.
- **detrend.first** Processed ts object
prep.check.acf

means Vector of means used in "sfsm" detrending method.
lambda Coefficient used in "boxcox" transformation.
start Start of the original time serie.
length Length of the original time serie.

Author(s)
Alberto Vico Moreno

References
https://www.otexts.org/fpp/8/1

See Also
prep.homogenize.log, prep.homogenize.boxcox, prep.detrend.differencing, prep.detrend.sfsm, prep.deseason.differencing, prep.check.acf, prep.check.adf

Examples
prep(AirPassengers)
prep(AirPassengers, homogenize.method='boxcox', detrend.method='none')

prep.check.acf Autocorrelation function

Description
Plots the autocorrelation function to check stationarity

Usage
prep.check.acf(tserie)

Arguments
tserie a ts or a prep object

Details
For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Also, for non-stationary data, the value is often large and positive.

Examples
prep.check.acf(AirPassengers)
prep.check.acf(prep(AirPassengers))
**prep.check.adf**

*Augmented Dickey-Fuller test*

**Description**

Performs ADF test just as another tool to check stationarity.

**Usage**

```r
prep.check.adf(tserie)
```

**Arguments**

- `tserie` a `ts` or a `prep` object

**Details**

Shows the results of an ADF test. A p-value<0.05 suggests the data is stationary.

**Examples**

```r
prep.check.adf(AirPassengers)
prep.check.adf(prep(AirPassengers))
```

---

**prep.deseason.differencing**

*Deseason with differencing method*

**Description**

Performs differencing with lag=frequency.

**Usage**

```r
prep.deseason.differencing(tserie, nsd = NULL)
```

**Arguments**

- `tserie` a `ts` object
- `nsd` number of seasonal differences to apply. As default its value is NULL; in this case, the function will perform an automatic estimation of `nsd`.

**Details**

If no number of differences is specified, the function will make an estimation of the number of differences needed based on unit root test provided by `forecast::nsiffs`
### prep.detrend.differencing

**Detrend with differencing method**

**Value**

A list is returned containing:

- **tserie**: Transformed ts object.
- **nsd**: Number of seasonal differences applied.
- **firstseasons**: Lost values after differencing.

**Examples**

```r
preg.detrend.differencing(AirPassengers)
preg.detrend.differencing(AirPassengers, nsd=2)
```

**Description**

Performs differencing with lag=1.

**Usage**

```r
prep.detrend.differencing(tserie, nd = NULL)
```

**Arguments**

- **tserie**: a ts object
- **nd**: number of differences to apply. As default its value is NULL; in this case, the function will perform an automatic estimation of nd.

**Details**

If no number of differences is specified, the function will make an estimation of the number of differences needed based on unit root test provided by forecast::ndiffs

**Value**

A list is returned containing:

- **tserie**: Transformed ts object.
- **nd**: Number of differences applied.
- **firstseasons**: Lost values after differencing.

**Examples**

```r
prep.detrend.differencing(AirPassengers)
preg.detrend.differencing(AirPassengers, nd=2)
```
**prep.detrend.sfsm**

**Detrend with "substracting full-season means" method**

**Description**

Performs "substracting full-season means" method to go for a totally automatic approach.

**Usage**

```r
prep.detrend.sfsm(tserie)
```

**Arguments**

- **tserie**: a ts object

**Details**

Under this detrending scheme, a series is first split into segments. The length of the segments is equal to the length of seasonality (12 for monthly). The mean of the historical observations within each of these segments is subtracted from every historical observation in the segment. To get the detrended serie we do: \( ds = x_i - m \) Being \( x_i \) the actual values on the time series and \( m \) the mean of the segment of \( x_i \)

**Value**

A list is returned containing:

- **tserie**: Transformed ts object.
- **means**: Vector containing the historical means.

**Examples**

```r
prep.detrend.sfsm(AirPassengers)
```

---

**prep.homogenize.boxcox**

*Box-Cox transformation*

**Description**

Performs a Box-Cox transformation to a time serie.

**Usage**

```r
prep.homogenize.boxcox(tserie)
```
Arguments

tserie  a ts object

Value

A list is returned containing:

<table>
<thead>
<tr>
<th>boxcox</th>
<th>Transformed ts object.</th>
</tr>
</thead>
<tbody>
<tr>
<td>lambda</td>
<td>Lambda value.</td>
</tr>
</tbody>
</table>

References

Box-Cox transformation: [https://en.wikipedia.org/wiki/Power_transform#Box-Cox_transformation](https://en.wikipedia.org/wiki/Power_transform#Box-Cox_transformation)

Examples

prep.homogenize.log(AirPassengers)
**print.modl**

*Generic function*

**Description**

Prints object `modl`

**Usage**

```r
## S3 method for class 'modl'
print(x, ...)
```

**Arguments**

- `x` prep object
- `...` ignored

**Examples**

```r
print(modl(prep(AirPassengers)))
```

---

**print.pred**

*Generic function*

**Description**

Prints object `pred`

**Usage**

```r
## S3 method for class 'pred'
print(x, ...)
```

**Arguments**

- `x` prep object
- `...` ignored

**Examples**

```r
print(pred(modl(prep(AirPassengers))))
```
print.prep

Generic function

Description

Prints object prep

Usage

## S3 method for class 'prep'
print(x, ...)

Arguments

x prep object
...

Arguments

... ignored

Examples

print(prep(AirPassengers))

summary.modl

Generic function

Description

Summary of object modl

Usage

## S3 method for class 'modl'
summary(object, ...)

Arguments

object prep object
...

Arguments

... ignored

Examples

summary(modl(prep(AirPassengers)))
**summary.pred**

Generic function

**Description**

Summary of object pred

**Usage**

```r
## S3 method for class 'pred'
summary(object, ...)
```

**Arguments**

- `object`: prep object
- `...`: ignored

**Examples**

```r
summary(pred(modl(prep(AirPassengers))))
```

**summary.prep**

Generic function

**Description**

Summary of object prep

**Usage**

```r
## S3 method for class 'prep'
summary(object, ...)
```

**Arguments**

- `object`: prep object
- `...`: ignored

**Examples**

```r
summary(prep(AirPassengers))
```
Index

modl, 2
modl.arima, 4
modl.dataMining, 4
modl.trControl, 4
modl.tsToDataFrame, 4

plot.pred, 7
plot.prep, 8
postp, 9
postp.deseason.differencing, 9
postp.detrend.differencing, 9
postp.detrend.sfsm, 9
postp.homogenize.boxcox, 9
postp.homogenize.log, 9
pred, 9
pred.arima, 13
pred.compareModels, 13
pred.dataMining, 13
prep, 4
prep.check.acf, 18
prep.check.adf, 18
prep.deseason.differencing, 18
prep.detrend.differencing, 18
prep.detrend.sfsm, 18
prep.homogenize.boxcox, 18
prep.homogenize.log, 18
print.modl, 23
print.pred, 23
print.prep, 24

summary.modl, 24
summary.pred, 25
summary.prep, 25