Package ‘LPStimeSeries’

March 27, 2015

Version 1.0-5
Date 2015-03-27
Title Learned Pattern Similarity and Representation for Time Series
Author Learned Pattern Similarity (LPS) for time series by Mustafa Gokce Baydogan
Depends R (>= 2.5.0)
Imports RColorBrewer
Maintainer Mustafa Gokce Baydogan <baydoganmustafa@gmail.com>
Description Learned Pattern Similarity (LPS) for time series.
   Implements a novel approach to model the dependency structure
   in time series that generalizes the concept of autoregression to local
   auto-patterns. Generates a pattern-based representation of time series
   along with a similarity measure called Learned Pattern Similarity (LPS).
   Introduces a generalized autoregressive kernel. This package is based on the
   ‘randomForest’ package by Andy Liaw.
License GPL (>= 2)
URL http://www.mustafabaydogan.com/learned-pattern-similarity-lps.html
NeedsCompilation yes
Repository CRAN
Date/Publication 2015-03-27 18:54:54

R topics documented:

  computeSimilarity . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
  getTreeInfo . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  GunPoint . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
  learnPattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  LPSNews . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  plot.learnPattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
  plotMDS . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11
  predict.learnPattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  tune.learnPattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14
  visualizePattern . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
computeSimilarity

Description
Compute similarity between time series. Raw time series can be provided together with learnPattern object so that the representation for the time series are generated internally and similarity is computed based on these representations. The other option is to provide the representations (instead of raw time series) and to compute the similarity without a need for learnPattern object.

Usage

computeSimilarity(object=NULL,testseries=NULL,refseries=NULL,
maxdepth=NULL,which.tree=NULL,sim.type=0, terminal=TRUE,
testrepresentation,refrepresentation)

Arguments

- **object**: an object of class learnPattern.
- **refseries**: reference time series.
- **testseries**: test time series.
- **maxdepth**: maximum depth level to be used to generate representations for similarity computations.
- **which.tree**: array of trees to be used for similarity computation.
- **sim.type**: type of the similarity to compute. If set to zero, dissimilarity (absolute differences of the number of patterns) is computed. If set to one, similarity (minimum number of the matching patterns) is computed.
- **terminal**: TRUE if similarity is computed over the learned representations.
- **testrepresentation**: learned representation for test time series.
- **refrepresentation**: learned representation for reference time series.

Value
A similarity matrix of size “the number of test series” by “the number of reference series“ is returned. Similarity between test series and reference series is defined as the number of mismatching patterns based on the representation generated by the trees. See LPS paper for details.

Note
Similarity matrix can also be computed over representations if it is generated using predict.learnPattern. This will probably take longer time compared to computing the similarity directly using the ensemble. However, if you are using LPS for retrieval purposes, bounding schemes (such as early abandon) can be used (requires further implementation) with the learned representations.
getTreeInfo

**Author(s)**

Mustafa Gokce Baydogan

**References**


**See Also**

`learnPattern, predict.learnPattern`

**Examples**

```r
data(GunPoint)
set.seed(71)
## Learn patterns on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries)

## Find the similarity between test and training series
sim=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries)

## Find similarity using representations,
## First generate representations
trainRep=predict(ensemble, GunPoint$trainseries, nodes=TRUE)
testRep=predict(ensemble, GunPoint$testseries, nodes=TRUE)

## Then compute the similarity (city-block distance),
## takes longer but we keep the representation
sim2=computeSimilarity(testrepresentation=testRep,refrepresentation=trainRep)

## Find the similarity based on first 100 trees
sim=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries,which.tree=c(1:100))
```

---

**getTreeInfo**

Extract a single tree from the ensemble.

**Description**

This function extracts the structure of a tree from a `learnPattern` object.

**Usage**

```r
getTreeInfo(object, which.tree=1)
```
getTreeInfo

Arguments

object a learnPattern object.
which.tree which tree to extract?

Value

is a list with the following components:

segment.length the proportion of the time series length used for both predictors and targets.
target starting time of the target segment.
target.type type of the target segment; 1 if observed series, 2 if difference series.
tree Tree structure matrix with seven columns and number of rows equal to total
number of nodes in the tree.

The seven columns of the tree structure matrix are:

left daughter the row where the left daughter node is; 0 if the node is terminal
right daughter the row where the right daughter node is; 0 if the node is terminal
split segment start time of the segment used to split the node
split type type of the predictor segment used to split the node; 1 if observed series, 2 if the
different series are used. 0 if the node is terminal
split point where the best split is
status is the node terminal (-1) or not (-3)
depth the depth of the node
prediction the prediction for the node

Note

For numerical predictors, data with values of the variable less than or equal to the splitting point go
to the left daughter node.

Author(s)

Mustafa Gokce Baydogan

See Also

learnPattern

Examples

data(GunPoint)
set.seed(71)

## Learn patterns on GunPoint training series with 50 trees
ensemble=learnPattern(GunPoint$trainseries,ntree=50)
getTreeInfo(ensemble, 3)
**GunPoint**

*The Gun-Point Data*

**Description**

This is the Gun-Point data from The UCR Time Series Database.

**Usage**

```r
data(GunPoint)
```

**Format**

GunPoint is a list with one training time series dataset and one test time series dataset provided as separate matrices. There are 50 cases (rows) for training dataset with 150 variables (columns). Similarly there are 150 cases for test dataset with 150 variables. Variables are representing the observations over time. In other words, they are ordered so that a row is a univariate time series. Originally, this is a classification problem where there are two classes. Therefore, list stores the class information for both training and test time series. This information is stored in arrays of length 50 and 150 for training and test time series respectively (so each time series is associated with a class).

Description by Chotirat Ann Ratanamahatana and Eamonn Keogh in their publication “Everything you know about Dynamic Time Warping is Wrong” is as follows:

“...This dataset comes from the video surveillance domain. The dataset has two classes, each containing 100 instances. All instances were created using one female actor and one male actor in a single session. The two classes are: Gun-Draw: The actors have their hands by their sides. They draw a replicate gun from a hip-mounted holster, point it at a target for approximately one second, then return the gun to the holster, and their hands to their sides. Point: The actors have their gun by their sides. They point with their index fingers to a target for approximately one second, and then return their hands to their sides. For both classes, we tracked the centroid of the actor’s right hands in both X- and Y-axes, which appear to be highly correlated; therefore, in this experiment, we only consider the X-axis for simplicity...“

**Author(s)**

Mustafa Gokce Baydogan

**Source**


**References**

**Learn Local Auto-Patterns for Time Series Representation and Similarity**

**Description**

`learnPattern` implements ensemble of regression trees (based on Breiman and Cutler’s original Fortran code) to learn local auto-patterns for time series representation. Ensemble of regression trees are used to learn an autoregressive model. A local time-varying autoregressive behavior is learned by the ensemble.

**Usage**

```r
## Default S3 method:
learnPattern(x, 
  segment.factor=c(0.05,0.95), 
  random.seg=TRUE, target.diff=TRUE, segment.diff=TRUE, 
  random.split=0, 
  ntree=200, 
  mtry=1, 
  replace=FALSE, 
  sampsize=if (replace) ceiling(0.632*nrow(x)) else nrow(x), 
  maxdepth=6, 
  nodesize=5, 
  do.trace=FALSE, 
  keep.forest=TRUE, 
```

See Also

- `learnPattern`, `computeSimilarity`

Examples

```r
data(GunPoint)
set.seed(71)

## Learn patterns on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries)
print(ensemble)

## Find the similarity between test and training series based on the learned model
similarity=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries)

## Find the index of 1 nearest neighbor (1NN) training series for each test series
NearestNeighbor=apply(similarity,1,which.min)

## Predicted class for each test series
predicted=GunPoint$trainclass[NearestNeighbor]
predicted
```
learnPattern

```r
oob.prd=FALSE,
keep.errors=FALSE,
keep.inbag=FALSE, ...)
## S3 method for class 'learnPattern'
print(x, ...)
```

### Arguments

- **x**: time series database as a matrix in UCR format. Rows are univariate time series, columns are observations (for the print method, a learnPattern object).

- **segment.factor**: The proportion of the time series length to be used for both predictors and targets, if `random.seg` is `TRUE` (default), minimum and maximum factor should be provided as array of length two.

- **random.seg**: `TRUE` if segment length is random between thresholds defined by `segment.factor`

- **target.diff**: Can target segment be a difference feature?

- **segment.diff**: Can predictor segments be difference feature?

- **random.split**: Type of the split. If set to zero (0), splits are generated based on decrease in SSE in target segment Setting of one (1) generates the split value randomly between max and min values. Setting of two (2) generates a kd-tree type of split (i.e. median of the values at each node is chosen as the split).

- **ntree**: Number of trees to grow. Larger number of trees are preferred if there is no concern regarding the computation time.

- **mtry**: Number of predictor segments randomly sampled as candidates at each split. Note that it is preset to 1 for now.

- **replace**: Should bagging of time series be done with replacement? All training time series are used if `FALSE` (default).

- **sampsize**: Size(s) of sample to draw with replacement if replace is set to `TRUE`

- **maxdepth**: The maximum depth of the trees in the ensemble.

- **nodesize**: Minimum size of terminal nodes. Setting this number larger causes smaller trees to be grown (and thus take less time).

- **do.trace**: If set to `TRUE`, give a more verbose output as learnPattern is run. If set to some integer, then running output is printed for every `do.trace` trees.

- **keep.forest**: If set to `FALSE`, the forest will not be retained in the output object.

- **oob.pred**: if `replace` is set to `TRUE`, predictions for the time series observations are returned.

- **keep.errors**: If set to `TRUE`, the mean square error (MSE) of target prediction over target segments is evaluated for each tree. If `oob.prd=TRUE`, this information is evaluated on “out-of-bag” samples at each tree.

- **keep.inbag**: Should an n by ntree matrix be returned that keeps track of which samples are “in-bag” in which trees

- **...**: optional parameters to be passed to the low level function learnPattern.
An object of class `learnPattern`, which is a list with the following components:

- **call**: the original call to `learnPattern`.
- **type**: regression
- **segment.factor**: the proportion of the time series length to be used for both predictors and targets.
- **segment.length**: used segment length settings by the trees of ensemble
- **nobs**: number of observations in a segment
- **ntree**: number of trees grown
- **maxdepth**: maximum depth level for each tree
- **mtry**: number of predictor segments sampled for splitting at each node.
- **target**: starting time of the target segment for each tree.
- **target.type**: type of the target segment; 1 if observed series, 2 if difference series.
- **forest**: a list that contains the entire forest; NULL if `keep.inbag` = FALSE.
- **oobprediction**: predicted observations based on “out-of-bag” time series are returned if `oob.pred` = TRUE
- **ooberrors**: Mean square error (MSE) over the trees evaluated using the predicted observations on “out-of-bag” time series is returned if `oob.pred` = TRUE.
- **inbag**: n by ntree matrix be returned that keeps track of which samples are “in-bag” in which trees if `keep.inbag` = TRUE
- **errors**: Mean square error (MSE) of target prediction over target segments for each tree. If `oob.pred` = TRUE, Mean square error (MSE) is reported based on “out-of-bag” samples at each tree.

Note

OOB predictions may have missing values (i.e. NA) if time series is not left out-of-bag during computations. Even, it is left out-of-bag, there is a potential of some observations (i.e. time frames) not being selected as the target. In such cases, there will no OOB predictions.

Author(s)

Mustafa Gokce Baydogan <baydogan.mustafa@gmail.com>, based on original Fortran code by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener.

References


See Also

`predict.learnPattern`, `computeSimilarity`, `tunelearnPattern`
Examples

data(GunPoint)
set.seed(71)

## Learn patterns on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries)
print(ensemble)

## Find the similarity between test and training series based on the learned model
similarity=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries)

## Find the index of 1 nearest neighbor (1NN) training series for each test series
NearestNeighbor=apply(similarity,1,which.min)

## Predicted class for each test series
predicted=GunPoint$trainclass[NearestNeighbor]

## Compute the percentage of accurate predictions
accuracy=sum(predicted==GunPoint$testclass)/nrow(GunPoint$testseries)
print(100*accuracy)

## Learn patterns randomly on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries, random.split=1)

## Find the similarity between test and training series and classify test series
similarity=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries)
NearestNeighbor=apply(similarity,1,which.min)
predicted=GunPoint$trainclass[NearestNeighbor]
accuracy=sum(predicted==GunPoint$testclass)/nrow(GunPoint$testseries)
print(100*accuracy)

## Learn patterns by training each tree on a random subsample
## and classify test time series
ensemble=learnPattern(GunPoint$trainseries,replace=TRUE)
similarity=computeSimilarity(ensemble,GunPoint$testseries,GunPoint$trainseries)
NearestNeighbor=apply(similarity,1,which.min)
predicted=GunPoint$trainclass[NearestNeighbor]
print(predicted)

## Learn patterns and do predictions on OOB time series
ensemble=learnPattern(GunPoint$trainseries,replace=TRUE,target.diff=FALSE,oob.pred=TRUE)

## Plot first series and its OOB approximation
plot(GunPoint$trainseries[1,],xlab='Time',ylab='Observation',
type='l',lty=1,lwd=2)
points(c(1:ncol(GunPoint$trainseries)),ensemble$oobpredictions[1,],
type='l',col=2,lty=2,lwd=2)
legend('topleft',c('Original series','Approximation'),
col=c(1,2),lty=c(1,2),lwd=2)
LPSNews

Show the NEWS file

Description

Show the NEWS file of the LPStimeSeries package.

Usage

LPSNews()

Value

None.

plot.learnPattern

Plot method for learnPattern objects

Description

Plot the MSE of a learnPattern object over trees based on out-of-bag predictions.

Usage

## S3 method for class 'learnPattern'
plot(x, type="l", main=deparse(substitute(x)), ...)

Arguments

x

an object of class learnPattern.

type
type of plot.

main

main title of the plot.

...

other graphical parameters.

Value

Invisibly, MSE of the learnPattern object.

Note

This function does not work for learnPattern if oob.predict=FALSE during training.

Author(s)

Mustafa Gokce Baydogan
See Also

`learnPattern`

Examples

```r
data(GunPoint)
ensemble=learnPattern(GunPoint$trainseries, oob.pred=TRUE, replace=TRUE)
plot(ensemble)
```

Plot the scaling coordinates of the Learned Pattern Similarity.

### Usage

```r
plotMDS(object, newdata, classinfo, k=2, palette=NULL, pch=20, ...)
```

### Arguments

- `object`: an object of class `learnPattern`, as that created by the function `learnPattern`.
- `newdata`: a data frame or matrix containing the data for similarity computation.
- `classinfo`: labels for the time series for color-coding.
- `k`: number of dimensions for the scaling coordinates.
- `palette`: colors to use to distinguish the classes; length must be the equal to the number of levels.
- `pch`: plotting symbols to use.
- `...`: other graphical parameters.

### Value

The output of `cmdscale` on scaled Learned Pattern similarity is returned invisibly.

### Note

If `k > 2`, `pairs` is used to produce the scatterplot matrix of the coordinates.

The entries of the similarity matrix is divided by the maximum possible similarity which is `2*sum(object$obs)`.

### Author(s)

Mustafa Gokce Baydogan
predict.learnpattern

See Also

learnpattern

Examples

```r
set.seed(1)
data(GunPoint)
## Learn patterns on GunPoint training series with default parameters
ensemble=learnpattern(GunPoint$trainseries)
plotMDS(ensemble, GunPoint$trainseries,GunPoint$trainclass)

## Using different symbols for the classes:
plotMDS(ensemble, GunPoint$trainseries,GunPoint$trainclass,
       palette=rep(1, 2), pch=as.numeric(GunPoint$trainclass))

## Learn patterns on GunPoint training series with random splits
ensemble=learnpattern(GunPoint$trainseries,random.split=1)
plotMDS(ensemble, GunPoint$trainseries,GunPoint$trainclass,main='Random Splits')
```

predict.learnpattern  predict method for learnPattern objects

Description

Representation generation for test data using learnPattern.

Usage

```r
## S3 method for class 'learnPattern'
predict(object, newdata, which.tree=NULL,
         nodes=TRUE, maxdepth=NULL, ...)
```

Arguments

- **object**: an object of class learnPattern, as that created by the function learnPattern.
- **newdata**: a data frame or matrix containing new data.
- **which.tree**: NULL if the representation is needed to be generated over all trees of ensemble. Set to an integer value if the representation is required to be generated for one tree specified by the value set.
- **nodes**: TRUE generates the representation based on the trees. FALSE generates a real-valued prediction for each time point.
- **maxdepth**: The maximum depth level to generate the representation
- **...**: not used currently.
Value

Returns the learned pattern representation for the time series in the dataset if nodes is set TRUE. Basically, it is the count of observed patterns at each terminal node. Otherwise predicted values for each time series in newdata are returned.

Author(s)

Mustafa Gokce Baydogan

References


See Also

learnPattern

Examples

data(GunPoint)
set.seed(71)
## Learn patterns on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries)

## Find representations
trainRep=predict(ensemble, GunPoint$trainseries, nodes=TRUE)
testRep=predict(ensemble, GunPoint$testseries, nodes=TRUE)

## Check size of the representation for training data
print(dim(trainRep))

## Learn patterns on GunPoint training series (target cannot be difference series)
ensemble=learnPattern(GunPoint$trainseries,target.diff=FALSE)

## Predict observations for test time series
predicted=predict(ensemble,GunPoint$testseries, nodes=FALSE)

## Plot an example test time series
plot(GunPoint$testseries[5,],type='l',lty=1,xlab='Time',ylab='Observation',lwd=2)
points(c(1:ncol(GunPoint$testseries)),predicted$predictions[5,],type='l',col=2,lty=2,lwd=2)
legend('topleft',c('Original series','Approximation'),col=c(1,2),lty=c(1,2),lwd=2)


tunelearnPattern  Tune Parameters of LPS for Time Series Classification

Description

tunelearnPattern implements parameter selection for LPS in time series classification problems. LPS requires the setting of segment length (if segment length is not random) and depth parameter. Given training time series and alternative parameter settings, the best set of parameters that minimizes the cross-validation error rate is returned.

Usage

tunelearnPattern(x, y, unlabeledx=NULL, nfolds=5,  
segmentlevels=c(0.25, 0.5, 0.75), random.split=0,  
mindepth=4, maxdepth=8, depthstep=2,  
ntreeTry=25, target.diff=TRUE, segment.diff=TRUE, ...)

Arguments

x  time series database as a matrix in UCR format. Rows are univariate time series, columns are observations (for the print method, a learnPattern object).
y  labels for the time series given by x
unlabeledx  unlabeled time series dataset. Introduced for future purposes as LPS can benefit from unlabeled data.
nfolds  number of cross-validation folds for parameter evaluation.
segmentlevels  alternative segment level settings to be evaluated. Settings are provided as an array.
random.split  Type of the split. If set to zero (0), splits are generated based on decrease in SSE in target segment. Setting of one (1) generates the split value randomly between max and min values. Setting of two (2) generates a kd-tree type of split (i.e. median of the values at each node is chosen as the split).
mindepth  minimum depth level to be evaluated.
maxdepth  maximum depth level to be evaluated.
depthstep  step size to determine the depth levels between mindepth and maxdepth to be evaluated.
ntreeTry  number of trees to be train for each fold.
target.diff  Can target segment be a difference feature?
segment.diff  Can predictor segments be difference feature?
...  optional parameters to be passed to the low level function tunelearnPattern.
**tunelearnPattern**

**Value**

A list with the following components:

- `params`: evaluated parameter combinations as a matrix where rows are parameter combinations and columns represent the settings. First and seconds columns are the evaluated segment length level and depth respectively.
- `errors`: cross-validation error rate for each parameter combinations
- `best.error`: the minimum cross-validation error rate obtained.
- `best.seg`: the segment length level that provides the minimum cross-validation error.
- `best.depth`: the depth level that provides the minimum cross-validation error.
- `random.split`: split type used for learning patterns.

**Author(s)**

Mustafa Gokce Baydogan <baydoganmustafa@gmail.com>, based on original Fortran code by Leo Breiman and Adele Cutler, R port by Andy Liaw and Matthew Wiener.

**References**


**See Also**

learnPattern, computeSimilarity

**Examples**

data(GunPoint)
set.seed(71)

```r
## Tune segment length level and depth on GunPoint training series
tuned = tunelearnPattern(GunPoint$trainseries, GunPoint$trainclass)
print(tuned$best.error)
print(tuned$best.seg)
print(tuned$best.depth)

## Use tuned parameters to learn patterns
ensemble = learnPattern(GunPoint$trainseries, segment.factor = tuned$best.seg, maxdepth = tuned$best.depth)

## Find the similarity between test and training series based on the learned model
similarity = computeSimilarity(ensemble, GunPoint@testseries, GunPoint$trainseries)

## Find the index of 1 nearest neighbor (1NN) training series for each test series
NearestNeighbor = apply(similarity, 1, which.min)

## Predicted class for each test series
```
predicted=GunPoint$trainclass[NearestNeighbor]

## Compute the percentage of accurate predictions
accuracy=sum(predicted==GunPoint$testclass)/nrow(GunPoint$testseries)
print(100*accuracy)

---

**visualizePattern**  
*Plot of the patterns learned by the ensemble of the regression trees*

**Description**

*visualizePattern* visualizes the patterns implied by the terminal nodes of the trees from *learnPattern* object.

**Usage**

```r
visualizePattern(object, x, which.terminal, orient=c(2,2))
```

**Arguments**

- **object**: an object of class *learnPattern*, as that created by the function *learnPattern*.
- **x**: a data frame or matrix containing the data for pattern visualization.
- **which.terminal**: id of the terminal node determining the decision rules to be used for identifying patterns.
- **orient**: orientation of the plot (determines the grid structure and how many patterns to be visualized).

**Value**

A list with the following components are returned invisibly.

- **predictor**: predictor segments residing in the which.terminal.
- **target**: target segments implied by the which.terminal.
- **tree**: the tree id corresponding to the which.terminal.
- **terminal**: the id of the terminal node for the tree.

**Note**

Patterns are visualized for the time series for which the frequency of the observations in the pattern is the largest. If more than one plot is requested through the setting of orient, the patterns are plotted for the time series based on the descending order of the frequency.

Currently, patterns are visualized based on the first predictor segment (sampled at the root node). This visualization can be done based on the predictor segment sampled at each level of the tree.

predictor and target are of size x where the patterns are numerical values and the rest of the entries are NAs.
**Author(s)**

Mustafa Gokce Baydogan

**See Also**

`learnPattern, predict.learnPattern`

**Examples**

```r
set.seed(71)
data(GunPoint)

## Learn patterns on GunPoint training series with default parameters
ensemble=learnPattern(GunPoint$trainseries)

## Find representations
trainRep=predict(ensemble, GunPoint$trainseries, nodes=TRUE)

## Find the average frequency over the terminal nodes
avgFreq=apply(trainRep,2,mean)

## Find the terminal node that has the maximum average and visualize
termid=which.max(avgFreq)
visualizePattern(ensemble,GunPoint$trainseries,termid,c(2,1))
```
Index

*Topic **classification**
  learnPattern, 6
  LPSNews, 10
  tunelearnPattern, 14

*Topic **datasets**
  GunPoint, 5

*Topic **regression**
  learnPattern, 6
  plot.learnPattern, 10
  tunelearnPattern, 14

*Topic **similarity**
  computeSimilarity, 2
  learnPattern, 6
  plotMDS, 11
  predict.learnPattern, 12
  visualizePattern, 16

*Topic **tree**
  computeSimilarity, 2
  getTreeInfo, 3
  learnPattern, 6
  plot.learnPattern, 10
  tunelearnPattern, 14

cmdscale, 11
computeSimilarity, 2, 6, 8, 15
getTreeInfo, 3
GunPoint, 5
learnPattern, 3, 4, 6, 6, 11–13, 15, 17
LPSNews, 10
pairs, 11
plot.learnPattern, 10
plotMDS, 11
predict.learnPattern, 2, 3, 8, 12, 17
print.learnPattern (learnPattern), 6

  tunelearnPattern, 8, 14

visualizePattern, 16