Package 'Modeler'

January 8, 2025

Version 3.4.7

Date 2025-01-08

Title Classes and Methods for Training and Using Binary Prediction Models

Depends R (>= 2.10), ClassDiscovery, ClassComparison, oompaBase

Imports methods, stats, class, rpart, TailRank, e1071, randomForest, nnet, neuralnet

Suggests Biobase

Description Defines classes and methods to learn models and use them to predict binary outcomes. These are generic tools, but we also include specific examples for many common classifiers.

License Apache License (== 2.0)

LazyLoad yes

biocViews Microarray, Clustering

URL http://oompa.r-forge.r-project.org/

NeedsCompilation no

Author Kevin R. Coombes [aut, cre]

Maintainer Kevin R. Coombes <krc@silicovore.com>

Repository CRAN

Date/Publication 2025-01-08 17:50:07 UTC

Contents

feature.filters 2
feature.selection 3
FittedModel 6
FittedModel-class 7
learn 8
learnCCP 9
learnKNN 11

learnLR 12
learnNNET 14
learnNNET2 16
learnPCALR 18
learnRF 20
learnRPART 22
learnSelectedLR 24
learnSVM 26
learnTailRank 28
Modeler 30
Modeler-class 32

Index **33**

`feature.filters` *Feature Filtering*

Description

Functions to create functions that filter potential predictive features using statistics that do not access class labels.

Usage

```
filterMean(cutoff)
filterMedian(cutoff)
filterSD(cutoff)
filterMin(cutoff)
filterMax(cutoff)
filterRange(cutoff)
filterIQR(cutoff)
```

Arguments

`cutoff`	A real number, the level above which features with this statistic should be retained and below which should be discarded.

Details

Following the usual conventions introduced from the world of gene expression microarrays, a typical data matrix is constructed from columns representing samples on which we want to make predictions amd rows representing the features used to construct the predictive model. In this context, we define a *filter* to be a function that accepts a data matrix as its only argument and returns a logical vector, whose length equals the number of rows in the matrix, where 'TRUE' indicates features that should be retrained. Most filtering functions belong to parametrized families, with one of the most common examples being "retain all features whose mean is above some pre-specified cutoff". We implement this idea using a set of function-generating functions, whose arguments are the parameters that pick out the desired member of the family. The return value is an instantiation

of a particular filtering function. The decison to define things this way is to be able to apply the methods in cross-validation (or other) loops where we want to ensure that we use the same filtering rule each time.

Value

Each of the seven functions described here return a filter function, `f`, that can be used by code that basically looks like `logicalVector <- filter(data)`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See `Modeler-class` and `Modeler` for details about how to train and test models.

Examples

```
set.seed(246391)
data <- matrix(rnorm(1000*30), nrow=1000, ncol=30)
fm <- filterMean(1)
summary(fm(data))

summary(filterMedian(1)(data))
summary(filterSD(1)(data))
```

---

feature.selection *Feature Selection*

Description

Functions to create functions that perform feature selection (or at least feature reduction) using statistics that access class labels.

Usage

```
keepAll(data, group)
fsTtest(fdr, ming=500)
fsModifiedFisher(q)
fsPearson(q = NULL, rho)
fsSpearman(q = NULL, rho)
fsMedSplitOddsRatio(q = NULL, OR)
fsChisquared(q = NULL, cutoff)
fsEntropy(q = 0.9, kind=c("information.gain", "gain.ratio", "symmetric.uncertainty"))
fsFisherRandomForest(q)
fsTailRank(specificity=0.9, tolerance=0.5, confidence=0.5)
```

Arguments

data	A matrix containng the data; columns are samples and rows are features.
group	A factor with two levels defining the sample classes.
fdr	A real number between 0 and 1 specifying the target false discovery rate (FDR).
ming	An integer specifing the minimum number of features to return; overrides the FDR.
q	A real number between 0.5 and 1 specifiying the fraction of features to discard.
rho	A real number between 0 and 1 specifying the absolute value of the correlation coefficient used to filter features.
OR	A real number specifying the desired odds ratio for filtering features.
cutoff	A real number specifiyng the targeted cutoff rate when using the statistic to filter features.
kind	The kind of information metric to use for filtering features.
specificity	See TailRankTest.
tolerance	See TailRankTest.
confidence	See TailRankTest.

Details

Following the usual conventions introduced from the world of gene expression microarrays, a typical data matrix is constructed from columns representing samples on which we want to make predictions amd rows representing the features used to construct the predictive model. In this context, we define a *feature selector* or *pruner* to be a function that accepts a data matrix and a two-level factor as its only arguments and returns a logical vector, whose length equals the number of rows in the matrix, where 'TRUE' indicates features that should be retrained. Most pruning functions belong to parametrized families. We implement this idea using a set of function-generating functions, whose arguments are the parameters that pick out the desired member of the family. The return value is an instantiation of a particular filtering function. The decison to define things this way is to be able to apply the methods in cross-validaiton (or other) loops where we want to ensure that we use the same feature selection rule each time.

We have implemented the following algorithms:

- keepAll: retain all features; do nothing.
- fsTtest: Keep features based on the false discovery rate from a two-goup t-test, but always retain a specified minimum number of genes.
- fsModifiedFisher Retain the top quantile of features for the statistic

$$\frac{(m_A - m)^2 + (m_B - m)^2}{v_A + v_B}$$

  where m is the mean and v is the variance.
- fsPearson: Retain the top quantile of features based on the absolute value of the Pearson correlation with the binary outcome.
- fsSpearman: Retain the top quantile of features based on the absolute value of the Spearman correlation with the binary outcome.

- `fsMedSplitOddsRatio`: Retain the top quantile of features based on the odds ratio to predict the binary outcome, after first dichotomizing the continuous predictor using a split at the median value.
- `fsChisquared`: retain the top quantile of features based on a chi-squared test comparing the binary outcome to continous predictors discretized into ten bins.
- `fsEntropy`: retain the top quantile of features based on one of three information-theoretic measures of entropy.
- `fsFisherRandomForest`: retain the top features based on their importance in a random forest analysis, after first filtering using the modified Fisher statistic.
- `fsTailRank`: Retain features that are significant based on the TailRank test, which is a measure of whether the tails of the distributions are different.

Value

The `keepAll` function is a "pruner"; it takes the data matrix and grouping factor as arguments, and returns a logical vector indicating which features to retain.

Each of the other nine functions described here return uses its arguments to contruct and return a pruning function, `f`, that has the same interface as `keepAll`.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See `Modeler-class` and `Modeler` for details about how to train and test models.

Examples

```
set.seed(246391)
data <- matrix(rnorm(1000*36), nrow=1000, ncol=36)
data[1:50, 1:18] <- data[1:50, 1:18] + 1
status <- factor(rep(c("A", "B"), each=18))

fsel <- fsPearson(q = 0.9)
summary(fsel(data, status))
fsel <- fsPearson(rho=0.3)
summary(fsel(data, status))

fsel <- fsEntropy(kind="gain.ratio")
summary(fsel(data, status))
```

FittedModel *Creating FittedModel objects*

Description

Construct an object of the FittedModel-class.

Usage

```
FittedModel(predict, data, status, details, ...)
```

Arguments

predict	A function that applies the model to predict outcomes on new test data.
data	A matrix containing the training data.
status	A vector containing the training outcomes, which should either be a binary-valued factor or a numeric vector of contiuous outcomes.
details	A list of the fitted parameters for the specified model.
...	Any extra information that is produced while learning the model; these wil be saved in the extras slot of the FittedModel object.

Details

Most users will never need to use this function; instead, they will first use an existing object of the Modeler-class, call the learn method of that object with the training data to obtain a FittedModel object, and then apply its predict method to test data. Only people who want to implement the learn-predict interface for a new classification algorithm are likely to need to call this function directly.

Value

Returns an object of the FittedModel-class.

Author(s)

Kevin R. Coombes <krc@silicovore.com.

See Also

See the descriptions of the learn function and the predict method for details on how to fit models on training data and make predictions on new test data.

See the description of the Modeler-class for details about the kinds of objects produced by learn.

Examples

```
# see the examples for learn and predict and for specific
# implementations of classifiers.
```

FittedModel-class *Class "FittedModel"*

Description

Objects of this class represent parametrized statistical models (of the Modeler-class) after they have been fit to a training data set. These objects can be used to predict binary outcomes on new test data sets.

Objects from the Class

Objects can be created by calls to the constructor function, FittedModel. In practice, however, most FittedModel objects are created as the result of applying the learn function to an object of the Modeler-class.

Slots

predictFunction: Object of class "function" that implemnts the ability to make predictions using the fitted model.

trainData: Object of class "matrix" containing the trainng data set. Rowes are features and columns are samples.

trainStatus: Object of class "vector". Should either be a numeric vector representing outcome or a factor with two levels, containing the classes of the training data set.

details: Object of class "list" containing the fitted parameters for the specific model.

extras: Object of class "list" containing any extra information (such as diagnostics) produced a a result of learning the model from the training data set.

fsVector: Logical vector indicating which features should be retained (TRUE) of discared (FALSE) after performing featgure selection on the training data.

Methods

predict signature(object = "FittedModel"): Predict the binary outcome on a new data set.

Author(s)

Kevin R. Coombes <krcoombes@mdanderson.org>

See Also

See Modeler-class and learn for details on how to fit a model to data.

Examples

```
showClass("FittedModel")
```

learn *Learning models from data*

Description

The learn function provides an abstraction that can be used to fit a binary classification model to a training data set.

Usage

```
learn(model, data, status, prune=keepAll)
```

Arguments

model	An object of the Modeler-class
data	A matrix containing the training data, with rows as features and columns as samples to be classified.
status	A factor, with two levels, containing the known classification of the training data.
prune	A "pruning" funciton; that is, a funciton that takes two arguments (a data matrix and a class factor) and returns a logical vector indicating which features to retain.

Details

Objects of the Modeler-class contain functions to learn models from training data to make predictions on new test data. These functions have to be prepared as pairs, since they have a shared opinion about how to record and use specific details about the parameters of the model. As a result, the learn function is implemented by:

```
learn <- function(model, data, status) {
  model@learn(data, status, model@params, model@predict)
}
```

Value

An object of the FittedModel-class.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See predict for how to make predictions on new test data from an object of the FittedModel-class.

Examples

```
# set up a generic RPART model
rpart.mod <- Modeler(learnRPART, predictRPART, minsplit=2, minbucket=1)

# simulate fake data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# learn the specific RPART model
fm <- learn(rpart.mod, data, status)

# show the predicted results from the model on the trianing data
predict(fm)

# set up a nearest neighbor model
knn.mod <- Modeler(learnKNN, predictKNN, k=3)

# fit the 3NN model on the same data
fm3 <- learn(knn.mod, data, status)
# show its performance
predict(fm3)
```

`learnCCP` *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic modeling mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnCCP(data, status, params, pfun)
predictCCP(newdata, details, status, ...)
```

Arguments

`data`	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
`status`	A factor, with two levels, classifying the samples. The length must equal the number of `data` columns.
`params`	A list of additional parameters used by the classifier; see Details.
`pfun`	The function used to make predictions on new data, using the trained classifier. Should always be set to `predictCCP`.
`newdata`	Another data matrix, with the same number of rows as `data`.
`details`	A list of additional parameters describing details about the particular classifier; see Details.
`...`	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnCCP and predictCCP are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The CCP classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnCCP function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnCCP is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictCCP function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnCCP function returns an object of the FittedModel-class, representing a CCP classifier that has been fitted on a training data set.

The predictCCP function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnCCP.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
ccp.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model
fm <- learnCCP(data, status, ccp.params, predictCCP)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictCCP(newdata, fm@details, status)
```

`learnKNN` *Fit models and make predictions with a KNN classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a K-nearest neighbors (KNN) classifier.

Usage

```
learnKNN(data, status, params, pfun)
predictKNN(newdata, details, status, ...)
```

Arguments

`data`	The data matrix, with rows as features and columns as the samples to be classified.
`status`	A factor, with two levels, classifying the samples. The length must equal the number of `data` columns.
`params`	A list of additional parameters used by the classifier; see Details.
`pfun`	The function used to make predictions on new data, using the trained classifier.
`newdata`	Another data matrix, with the same number of rows as `data`.
`details`	A list of additional parameters describing details about the particular classifier; see Details.
`...`	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both `learnKNN` and `predictKNN` are dictated by the requirements of the general train-and-test mechanism provided by the `Modeler-class`.

The implementation uses the `knn` method from the `class` package. The `params` argument to `learnKNN` must be alist that at least includes the component `k` that specifies the number of neighbors used.

Value

The learnKNN function returns an object of the FittedModel-class, logically representing a KNN classifier that has been fitted on a training data set.

The predictKNN function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

References

Ripley, B. D. (1996) *Pattern Recognition and Neural Networks*. Cambridge.

Venables, W. N. and Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth edition. Springer.

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnPCALR.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
knn.params <- list(k=5)

# learn the model
fm <- learnKNN(data, status, knn.params, predictKNN)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictKNN(newdata, fm@details, status)
```

---

learnLR *Fit models and make predictions with a logistic regression classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a logistic regression (LR) classifier.

Usage

```
learnLR(data, status, params, pfun)
predictLR(newdata, details, status, type ="response", ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictLR.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
type	A character string indicating the type of prediciton to make.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnLR and predictLR are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The LR classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnLR function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnLR is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictLR function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnLR function returns an object of the FittedModel-class, representing a LR classifier that has been fitted on a training data set.

The predictLR function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnLR.

Examples

```
## Not run:
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
lr.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model -- this is slow
fm <- learnLR(data, status, lr.params, predictLR)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictLR(newdata, fm@details, status)

## End(Not run)
```

---

learnNNET *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnNNET(data, status, params, pfun)
predictNNET(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictNNET.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnNNET and predictNNET are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The NNET classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnNNET function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnNNET is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictNNET function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnNNET function returns an object of the FittedModel-class, representing a NNET classifier that has been fitted on a training data set.

The predictNNET function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnNNET.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
nnet.params <- list()

# learn the model
#fm <- learnNNET(data, status, nnet.params, predictNNET)

# Make predictions on some new simulated data
#newdata <- matrix(rnorm(100*30), ncol=30)
#predictNNET(newdata, fm@details, status)
```

---

learnNNET2 *Fit models and make predictions with a multi-level neural network classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier usinfg neural networks.

Usage

```
learnNNET2(data, status, params, pfun)
predictNNET2(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictNNET2.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnNNET2 and predictNNET2 are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The NNET2 classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnNNET2 function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnNNET2 is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictNNET2 function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnNNET2 function returns an object of the FittedModel-class, representing a NNET2 classifier that has been fitted on a training data set.

The predictNNET2 function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnNNET2.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
nnet.params <- list()

# learn the model
#fm <- learnNNET2(data, status, nnet.params, predictNNET2)

# Make predictions on some new simulated data
#newdata <- matrix(rnorm(100*30), ncol=30)
#predictNNET2(newdata, fm@details, status)
```

---

learnPCALR *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnPCALR(data, status, params, pfun)
predictPCALR(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictPCALR.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnPCALR and predictPCALR are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The PCALR classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnPCALR function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnPCALR is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictPCALR function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnPCALR function returns an object of the FittedModel-class, representing a PCALR classifier that has been fitted on a training data set.

The predictPCALR function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnPCALR.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
pcalr.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model
fm <- learnPCALR(data, status, pcalr.params, predictPCALR)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictPCALR(newdata, fm@details, status)
```

---

learnRF *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnRF(data, status, params, pfun)
predictRF(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictRF.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnRF and predictRF are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The RF classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnRF function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnRF is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictRF function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnRF function returns an object of the FittedModel-class, representing a RF classifier that has been fitted on a training data set.

The predictRF function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnRF.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
svm.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model
#fm <- learnRF(data, status, svm.params, predictRF)

# Make predictions on some new simulated data
#newdata <- matrix(rnorm(100*30), ncol=30)
#predictRF(newdata, fm@details, status)
```

---

learnRPART *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnRPART(data, status, params, pfun)
predictRPART(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictRPART.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnRPART and predictRPART are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The RPART classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnRPART function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnRPART is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictRPART function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnRPART function returns an object of the FittedModel-class, representing a RPART classifier that has been fitted on a training data set.

The predictRPART function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnRPART.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
rpart.params <- list(minsplit = 10)

# learn the model
fm <- learnRPART(data, status, rpart.params, predictRPART)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictRPART(newdata, fm@details, status)
```

---

learnSelectedLR *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnSelectedLR(data, status, params, pfun)
predictSelectedLR(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictSelectedLR.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnSelectedLR and predictSelectedLR are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The SelectedLR classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnSelectedLR function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnSelectedLR is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictSelectedLR function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnSelectedLR function returns an object of the FittedModel-class, representing a SelectedLR classifier that has been fitted on a training data set.

The predictSelectedLR function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to tain and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnSelectedLR.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
slr.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model
fm <- learnSelectedLR(data, status, slr.params, predictSelectedLR)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictSelectedLR(newdata, fm@details, status)
```

---

learnSVM *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnSVM(data, status, params, pfun)
predictSVM(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictSVM.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnSVM and predictSVM are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The SVM classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnSVM function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnSVM is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictSVM function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The `learnSVM` function returns an object of the `FittedModel-class`, representing a SVM classifier that has been fitted on a training `data` set.

The `predictSVM` function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See `Modeler-class` and `Modeler` for details about how to train and test models. See `FittedModel-class` and `FittedModel` for details about the structure of the object returned by `learnSVM`.

Examples

```
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
svm.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model
fm <- learnSVM(data, status, svm.params, predictSVM)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictSVM(newdata, fm@details, status)
```

---

learnTailRank *Fit models and make predictions with a PCA-LR classifier*

Description

These functions are used to apply the generic train-and-test mechanism to a classifier that combines principal component analysis (PCA) with logistic regression (LR).

Usage

```
learnTailRank(data, status, params, pfun)
predictTailRank(newdata, details, status, ...)
```

Arguments

data	The data matrix, with rows as features ("genes") and columns as the samples to be classified.
status	A factor, with two levels, classifying the samples. The length must equal the number of data columns.
params	A list of additional parameters used by the classifier; see Details.
pfun	The function used to make predictions on new data, using the trained classifier. Should always be set to predictTailRank.
newdata	Another data matrix, with the same number of rows as data.
details	A list of additional parameters describing details about the particular classifier; see Details.
...	Optional extra parameters required by the generic "predict" method.

Details

The input arguments to both learnTailRank and predictTailRank are dictated by the requirements of the general train-and-test mechanism provided by the Modeler-class.

The TailRank classifier is similar in spirit to the "supervised principal components" method implemented in the superpc package. We start by performing univariate two-sample t-tests to identify features that are differentially expressed between two groups of training samples. We then set a cutoff to select features using a bound (alpha) on the false discovery rate (FDR). If the number of selected features is smaller than a prespecified goal (minNgenes), then we increase the FDR until we get the desired number of features. Next, we perform PCA on the selected features from the trqining data. we retain enough principal components (PCs) to explain a prespecified fraction of the variance (perVar). We then fit a logistic regression model using these PCs to predict the binary class of the training data. In order to use this model to make binary predictions, you must specify a prior probability that a sample belongs to the first of the two groups (where the ordering is determined by the levels of the classification factor, status).

In order to fit the model to data, the params argument to the learnTailRank function should be a list containing components named alpha, minNgenes, perVar, and prior. It may also contain a logical value called verbose, which controls the amount of information that is output as the algorithm runs.

The result of fitting the model using learnTailRank is a member of the FittedModel-class. In additon to storing the prediction function (pfun) and the training data and status, the FittedModel stores those details about the model that are required in order to make predictions of the outcome on new data. In this acse, the details are: the prior probability, the set of selected features (sel, a logical vector), the principal component decomposition (spca, an object of the SamplePCA class), the logistic regression model (mmod, of class glm), the number of PCs used (nCompUsed) as well as the number of components available (nCompAvail) and the number of gene-features selected (nGenesSelecets). The details object is appropriate for sending as the second argument to the predictTailRank function in order to make predictions with the model on new data. Note that the status vector here is the one used for the *training* data, since the prediction function only uses the *levels* of this factor to make sure that the direction of the predicitons is interpreted correctly.

Value

The learnTailRank function returns an object of the FittedModel-class, representing a TailRank classifier that has been fitted on a training data set.

The predictTailRank function returns a factor containing the predictions of the model when applied to the new data set.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See Modeler-class and Modeler for details about how to train and test models. See FittedModel-class and FittedModel for details about the structure of the object returned by learnTailRank.

Examples

```
## Not run:
# simulate some data
data <- matrix(rnorm(100*20), ncol=20)
status <- factor(rep(c("A", "B"), each=10))

# set up the parameter list
tr.params <- list(minNgenes=10, alpha=0.10, perVar=0.80, prior=0.5)

# learn the model -- this is slow
fm <- learnTailRank(data, status, tr.params, predictTailRank)

# Make predictions on some new simulated data
newdata <- matrix(rnorm(100*30), ncol=30)
predictTailRank(newdata, fm@details, status)

## End(Not run)
```

Modeler *Constructor for "Modeler" objects*

Description

The Modeler-class represents (parametrized but not yet fit) statistical models that can predict binary outcomes. The Modeler function is used to construct objects of this class.

Usage

```
Modeler(learn, predict, ...)
```

Arguments

learn	Object of class "function" that will be used to fit the model to a data set. See learn for details.
predict	Object of class "function" that will be used to make predictions on new data from a fitted model. See predict for details.
...	Additional parameters required for the specific kind of classificaiton model that will be constructed. See Details.

Details

Objects of the Modeler-class provide a general abstraction for classification models that can be learned from one data set and then applied to a new data set. Each type of classifier is likely to have its own specific parameters. For instance, a K-nearest neighbors classifier requires you to specify k. The more complex classifier, PCA-LR has many more parameters, including the false discovery rate (alpha) used to select features and the percentage of variance (perVar) that should be explained by the number of principal components created from those features. All additional parameters should be suplied as named arguments to the Modeler constructor; these additional parameters will be bundled into a list and inserted into the params slot of the resulting object of the Modeler-class.

Value

Returns an object of the Modeler-class.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See the descriptions of the learn function and the predict method for details on how to fit models on training data and make predictions on new test data.

See the description of the FittedModel-class for details about the kinds of objects produced by learn.

Examples

```
learnNNET
predictNNET
modelerNNET <- Modeler(learnNNET, predictNNET, size=5)
modelerNNET
```

Modeler-class *Class "Modeler"*

Description

The Modeler class represents (parametrized but not yet fit) statistical models that can predict binary outcomes.

Objects from the Class

Objects can be created by calls to the constructor fuinction, Modeler.

Slots

learnFunction: Object of class "function" that is used to fit the model to a data set. See learn for details.

predictFunction: Object of class "function" that is used to make predictions on new data from a fitted model. See predict for details.

paramList: Object of class "list" that contains parameters that are specific for one type of classifier.

Methods

No methods are defined with class "Modeler" in the signature. The only function that can be applied to a Modeler object is learn, which has not been made into a generic funtion.

Author(s)

Kevin R. Coombes <krc@silicovore.com>

See Also

See the description of the FittedModel-class for details about the kinds of objects produced by learn.

Examples

```
showClass("Modeler")
```

Index

∗ **classes**
FittedModel-class, 7
Modeler-class, 32
∗ **classif**
FittedModel, 6
learn, 8
learnCCP, 9
learnKNN, 11
learnLR, 12
learnNNET, 14
learnNNET2, 16
learnPCALR, 18
learnRF, 20
learnRPART, 22
learnSelectedLR, 24
learnSVM, 26
learnTailRank, 28
Modeler, 30
∗ **multivariate**
feature.filters, 2
feature.selection, 3
FittedModel, 6
learn, 8
learnCCP, 9
learnKNN, 11
learnLR, 12
learnNNET, 14
learnNNET2, 16
learnPCALR, 18
learnRF, 20
learnRPART, 22
learnSelectedLR, 24
learnSVM, 26
learnTailRank, 28
Modeler, 30

class, *11*

feature.filters, 2
feature.selection, 3

filterIQR *(*feature.filters*)*, 2
filterMax *(*feature.filters*)*, 2
filterMean *(*feature.filters*)*, 2
filterMedian *(*feature.filters*)*, 2
filterMin *(*feature.filters*)*, 2
filterRange *(*feature.filters*)*, 2
filterSD *(*feature.filters*)*, 2
FittedModel, 6, *7*, *10*, *12*, *14*, *16*, *18*, *20*, *22*, *24*, *26*, *28*, *30*
FittedModel-class, 7
fsChisquared *(*feature.selection*)*, 3
fsEntropy *(*feature.selection*)*, 3
fsFisherRandomForest *(*feature.selection*)*, 3
fsMedSplitOddsRatio *(*feature.selection*)*, 3
fsModifiedFisher *(*feature.selection*)*, 3
fsPearson *(*feature.selection*)*, 3
fsSpearman *(*feature.selection*)*, 3
fsTailRank *(*feature.selection*)*, 3
fsTtest *(*feature.selection*)*, 3

glm, *10*, *13*, *15*, *17*, *19*, *21*, *23*, *25*, *27*, *29*

keepAll *(*feature.selection*)*, 3
knn, *11*

learn, *6*, *7*, 8, *31*, *32*
learnCCP, 9
learnKNN, 11
learnLR, 12
learnNNET, 14
learnNNET2, 16
learnPCALR, 18
learnRF, 20
learnRPART, 22
learnSelectedLR, 24
learnSVM, 26
learnTailRank, 28

Modeler, *3*, *5*, *10*, *12*, *14*, *16*, *18*, *20*, *22*, *24*, *26*, *28*, *30*, 30, *32*
Modeler-class, 32
modeler3NN *(*learnKNN*)*, 11
modeler5NN *(*learnKNN*)*, 11
modelerCCP *(*learnCCP*)*, 9
modelerLR *(*learnLR*)*, 12
modelerNNET *(*learnNNET*)*, 14
modelerNNET2 *(*learnNNET2*)*, 16
modelerPCALR *(*learnPCALR*)*, 18
modelerRF *(*learnRF*)*, 20
modelerRPART *(*learnRPART*)*, 22
modelerSelectedLR *(*learnSelectedLR*)*, 24
modelerSVM *(*learnSVM*)*, 26
modelerTailRank *(*learnTailRank*)*, 28

predict, *6–8*, *31*, *32*
predict,FittedModel-method *(*FittedModel-class*)*, 7
predictCCP *(*learnCCP*)*, 9
predictKNN *(*learnKNN*)*, 11
predictLR *(*learnLR*)*, 12
predictNNET *(*learnNNET*)*, 14
predictNNET2 *(*learnNNET2*)*, 16
predictPCALR *(*learnPCALR*)*, 18
predictRF *(*learnRF*)*, 20
predictRPART *(*learnRPART*)*, 22
predictSelectedLR *(*learnSelectedLR*)*, 24
predictSVM *(*learnSVM*)*, 26
predictTailRank *(*learnTailRank*)*, 28

SamplePCA, *10*, *13*, *15*, *17*, *19*, *21*, *23*, *25*, *27*, *29*

TailRankTest, *4*