Package 'lolR'

June 27, 2020

Type Package

Title Linear Optimal Low-Rank Projection

Version 2.1

Date 2020-06-20

Maintainer Eric Bridgeford <ericwb95@gmail.com>

Description Supervised learning techniques designed for the situation when the dimensionality exceeds the sample size have a tendency to overfit as the dimensionality of the data increases. To remedy this High dimensionality; low sample size (HDLSS) situation, we attempt to learn a lower-dimensional representation of the data before learning a classifier. That is, we project the data to a situation where the dimensionality is more manageable, and then are able to better apply standard classification or clustering techniques since we will have fewer dimensions to overfit. A number of previous works have focused on how to strategically reduce dimensionality in the unsupervised case, yet in the supervised HDLSS regime, few works have attempted to devise dimensionality reduction techniques that leverage the labels associated with the data. In this package and the associated manuscript Vogelstein et al. (2017) <arXiv:1709.01233>, we provide several methods for feature extraction, some utilizing labels and some not, along with easily extensible utilities to simplify cross-validative efforts to identify the best feature extraction method. Additionally, we include a series of adaptable benchmark simulations to serve as a standard for future investigative efforts into supervised HDLSS. Finally, we produce a comprehensive comparison of the included algorithms across a range of benchmark simulations and real data applications.

Depends R (>= 3.4.0)

License GPL-2

URL https://github.com/neurodata/lol

Imports ggplot2, abind, MASS, irlba, pls, robust, robustbase

Encoding UTF-8

LazyData true

VignetteBuilder knitr

RoxygenNote 7.1.0

Suggests knitr, rmarkdown, parallel, randomForest, latex2exp, testthat, covr

NeedsCompilation no

Author Eric Bridgeford [aut, cre],
Minh Tang [ctb],
Jason Yim [ctb],
Joshua Vogelstein [ths]

Repository CRAN

Date/Publication 2020-06-26 22:30:03 UTC

R topics documented:

`lol.classify.nearestCentroid` *Nearest Centroid Classifier Training*

Description

A function that trains a classifier based on the nearest centroid.

Usage

```
lol.classify.nearestCentroid(X, Y, ...)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the `n` samples.
`...`	optional args.

Value

A list of class `nearestCentroid`, with the following attributes:

`centroids`	`[K,d]` the centroids of each class with `K` classes in `d` dimensions.
`ylabs`	`[K]` the ylabels for each of the `K` unique classes, ordered.
`priors`	`[K]` the priors for each of the `K` classes.

Details

For more details see the help vignette: `vignette("centroid",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.nearestCentroid(X, Y)
```

lol.classify.rand *Random Classifier Utility*

Description

A function for random classifiers.

Usage

```
lol.classify.rand(X, Y, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
Y	[n] the labels of the n samples.
...	optional args.

Value

A structure, with the following attributes:

ylabs	[K] the ylabels for each of the K unique classes, ordered.
priors	[K] the priors for each of the K classes.

Author(s)

Eric Bridgeford

lol.classify.randomChance *Randomly Chance Classifier Training*

Description

A function that predicts the maximally present class in the dataset. Functionality consistent with the standard R prediction interface so that one can compute the "chance" accuracy with minimal modification of other classification scripts.

Usage

```
lol.classify.randomChance(X, Y, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
Y	[n] the labels of the n samples.
...	optional args.

Value

A list of class randomGuess, with the following attributes:

ylabs	[K] the ylabels for each of the K unique classes, ordered.
priors	[K] the priors for each of the K classes.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.randomChance(X, Y)
```

---

lol.classify.randomGuess

Randomly Guessing Classifier Training

Description

A function that predicts by randomly guessing based on the pmf of the class priors. Functionality consistent with the standard R prediction interface so that one can compute the "guess" accuracy with minimal modification of other classification scripts.

Usage

```
lol.classify.randomGuess(X, Y, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
Y	[n] the labels of the n samples.
...	optional args.

Value

A list of class `randomGuess`, with the following attributes:

`ylabs`	`[K]` the ylabels for each of the `K` unique classes, ordered.
`priors`	`[K]` the priors for each of the `K` classes.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.randomGuess(X, Y)
```

`lol.embed` *Embedding*

Description

A function that embeds points in high dimensions to a lower dimensionality.

Usage

```
lol.embed(X, A, ...)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`A`	`[d,r]` the embedding matrix from `d` to `r` dimensions.
`...`	optional args.

Value

an array `[n,r]` the original `n` points embedded into `r` dimensions.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.lol(X=X, Y=Y, r=5)  # use lol to project into 5 dimensions
Xr <- lol.embed(X, model$A)
```

`lol.project.bayes_optimal` *Bayes Optimal*

Description

A function for recovering the Bayes Optimal Projection, which optimizes Bayes classification.

Usage

```
lol.project.bayes_optimal(X, Y, mus, Sigmas, priors, ...)
```

Arguments

`X`	`[n,p]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`...`	optional args.

Value

A list of class `embedding` containing the following:

`A`	`[d,K]` the projection matrix from `d` to `K` dimensions.
`d`	the eigen values associated with the eigendecomposition.
`ylabs`	`[K]` vector containing the `K` unique, ordered class labels.
`centroids`	`[K,d]` centroid matrix of the `K` unique, ordered classes in native `d` dimensions.
`priors`	`[K]` vector containing the `K` prior probabilities for the unique, ordered classes.
`Xr`	`[n,K]` the `n` data points in reduced dimensionality `K`.
`cr`	`[K,K]` the `K` centroids in reduced dimensionality `K`.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
# obtain bayes-optimal projection of the data
model <- lol.project.bayes_optimal(X=X, Y=Y, mus=data$mus,
                                   S=data$Sigmas, priors=data$priors)
```

`lol.project.dp` *Data Piling*

Description

A function for implementing the Maximal Data Piling (MDP) Algorithm.

Usage

```
lol.project.dp(X, Y, ...)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels.
`...`	optional args.

Value

A list containing the following:

`A`	`[d,r]` the projection matrix from `d` to `r` dimensions.
`ylabs`	`[K]` vector containing the `K` unique, ordered class labels.
`centroids`	`[K,d]` centroid matrix of the `K` unique, ordered classes in native `d` dimensions.
`priors`	`[K]` vector containing the `K` prior probabilities for the unique, ordered classes.
`Xr`	`[n,r]` the `n` data points in reduced dimensionality `r`.
`cr`	`[K,r]` the `K` centroids in reduced dimensionality `r`.

Details

For more details see the help vignette: `vignette("dp",package = "lolR")`

Author(s)

Minh Tang and Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.dp(X=X, Y=Y)  # use mdp to project into maximal data piling
```

`lol.project.lol` *Linear Optimal Low-Rank Projection (LOL)*

Description

A function for implementing the Linear Optimal Low-Rank Projection (LOL) Algorithm. This algorithm allows users to find an optimal projection from ‘d‘ to ‘r‘ dimensions, where ‘r « d‘, by combining information from the first and second moments in thet data.

Usage

```
lol.project.lol(
  X,
  Y,
  r,
  second.moment.xfm = FALSE,
  second.moment.xfm.opts = list(),
  first.moment = "delta",
  second.moment = "linear",
  orthogonalize = FALSE,
  robust = FALSE,
  ...
)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels.
`r`	the rank of the projection. Note that `r >= K`, and `r < d`.
`second.moment.xfm`	whether to use extraneous options in estimation of the second moment component. The transforms specified should be a numbered list of transforms you wish to apply, and will be applied in accordance with `second.moment`.
`second.moment.xfm.opts`	optional arguments to pass to the `second.moment.xfm` option specified. Should be a numbered list of lists, where `second.moment.xfm.opts[[i]]` corresponds to the optional arguments for `second.moment.xfm[[i]]`. Defaults to the default options for each transform scheme.
`first.moment`	the function to capture the first moment. Defaults to `'delta'`. • `'delta'` capture the first moment with the hyperplane separating the per-class means. • `FALSE` do not capture the first moment.
`second.moment`	the function to capture the second moment. Defaults to `'linear'`.

- 'linear' performs PCA on the class-conditional data to capture the second moment, retaining the vectors with the top singular values. Transform options for second.moment.xfm and arguments in second.moment.opts should be in accordance with the trailing arguments for lol.project.lrlda.
- 'quadratic' performs PCA on the data for each class separately to capture the second moment, retaining the vectors with the top singular values from each class's PCA. Transform options for second.moment.xfm and arguments in second.moment.opts should be in accordance with the trailing arguments for lol.project.pca.
- 'pls' performs PLS on the data to capture the second moment, retaining the vectors that maximize the correlation between the different classes. Transform options for second.moment.xfm and arguments in second.moment.opts should be in accordance with the trailing arguments for lol.project.pls.
- FALSE do not capture the second moment.

orthogonalize	whether to orthogonalize the projection matrix. Defaults to FALSE.
robust	whether to perform PCA on a robust estimate of the covariance matrix or not. Defaults to FALSE.
...	trailing args.

Value

A list containing the following:

A	[d,r] the projection matrix from d to r dimensions.
ylabs	[K] vector containing the K unique, ordered class labels.
centroids	[K,d] centroid matrix of the K unique, ordered classes in native d dimensions.
priors	[K] vector containing the K prior probabilities for the unique, ordered classes.
Xr	[n,r] the n data points in reduced dimensionality r.
cr	[K,r] the K centroids in reduced dimensionality r.
second.moment	the method used to estimate the second moment.
first.moment	the method used to estimate the first moment.

Details

For more details see the help vignette: vignette("lol",package = "lolR")

Author(s)

Eric Bridgeford

References

Joshua T. Vogelstein, et al. "Supervised Dimensionality Reduction for Big Data" arXiv (2020).

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.lol(X=X, Y=Y, r=5)  # use lol to project into 5 dimensions

# use lol to project into 5 dimensions, and produce an orthogonal basis for the projection matrix
model <- lol.project.lol(X=X, Y=Y, r=5, orthogonalize=TRUE)

# use LRQDA to estimate the second moment by performing PCA on each class
model <- lol.project.lol(X=X, Y=Y, r=5, second.moment='quadratic')


# use PLS to estimate the second moment
model <- lol.project.lol(X=X, Y=Y, r=5, second.moment='pls')

# use LRLDA to estimate the second moment, and apply a unit transformation
# (according to scale function) with no centering
model <- lol.project.lol(X=X, Y=Y, r=5, second.moment='linear', second.moment.xfm='unit',
                         second.moment.xfm.opts=list(center=FALSE))
```

---

lol.project.lrcca *Low-rank Canonical Correlation Analysis (LR-CCA)*

Description

A function for implementing the Low-rank Canonical Correlation Analysis (LR-CCA) Algorithm.

Usage

```
lol.project.lrcca(X, Y, r, ...)
```

Arguments

X	[n, d] the data with n samples in d dimensions.
Y	[n] the labels of the samples with K unique labels.
r	the rank of the projection.
...	trailing args.

Value

A list containing the following:

A	[d,r] the projection matrix from d to r dimensions.
d	the eigen values associated with the eigendecomposition.
ylabs	[K] vector containing the K unique, ordered class labels.
centroids	[K,d] centroid matrix of the K unique, ordered classes in native d dimensions.

priors	[K] vector containing the K prior probabilities for the unique, ordered classes.
Xr	[n,r] the n data points in reduced dimensionality r.
cr	[K,r] the K centroids in reduced dimensionality r.

Details

For more details see the help vignette: `vignette("lrcca",package = "lolR")`

Author(s)

Eric Bridgeford and Minh Tang

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.lrcca(X=X, Y=Y, r=5)  # use lrcca to project into 5 dimensions
```

---

lol.project.lrlda *Low-Rank Linear Discriminant Analysis (LRLDA)*

Description

A function that performs LRLDA on the class-centered data. Same as class-conditional PCA.

Usage

```
lol.project.lrlda(X, Y, r, xfm = FALSE, xfm.opts = list(), robust = FALSE, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
Y	[n] the labels of the samples with K unique labels.
r	the rank of the projection.
xfm	whether to transform the variables before taking the SVD.

- FALSEapply no transform to the variables.
- 'unit'unit transform the variables, defaulting to centering and scaling to mean 0, variance 1. See `scale` for details and optional args.
- 'log'log-transform the variables, for use-cases such as having high variance in larger values. Defaults to natural logarithm. See `log` for details and optional args.
- 'rank'rank-transform the variables. Defalts to breaking ties with the average rank of the tied values. See `rank` for details and optional args.
- c(opt1, opt2, etc.)apply the transform specified in opt1, followed by opt2, etc.

xfm.opts	optional arguments to pass to the xfm option specified. Should be a numbered list of lists, where xfm.opts[[i]] corresponds to the optional arguments for xfm[i]. Defaults to the default options for each transform scheme.
robust	whether to use a robust estimate of the covariance matrix when taking PCA. Defaults to FALSE.
...	trailing args.

Value

A list containing the following:

A	[d,r] the projection matrix from d to r dimensions.
d	the eigen values associated with the eigendecomposition.
ylabs	[K] vector containing the K unique, ordered class labels.
centroids	[K,d] centroid matrix of the K unique, ordered classes in native d dimensions.
priors	[K] vector containing the K prior probabilities for the unique, ordered classes.
Xr	[n,r] the n data points in reduced dimensionality r.
cr	[K,r] the K centroids in reduced dimensionality r.

Details

For more details see the help vignette: vignette("lrlda",package = "lolR")

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.lrlda(X=X, Y=Y, r=2)  # use lrlda to project into 2 dimensions
```

---

lol.project.pca *Principal Component Analysis (PCA)*

Description

A function that performs PCA on data.

Usage

```
lol.project.pca(X, r, xfm = FALSE, xfm.opts = list(), robust = FALSE, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
r	the rank of the projection.
xfm	whether to transform the variables before taking the SVD. • FALSEapply no transform to the variables. • 'unit'unit transform the variables, defaulting to centering and scaling to mean 0, variance 1. See scale for details and optional arguments to be passed with xfm.opts. • 'log'log-transform the variables, for use-cases such as having high variance in larger values. Defaults to natural logarithm. See log for details and optional arguments to be passed with xfm.opts. • 'rank'rank-transform the variables. Defalts to breaking ties with the average rank of the tied values. See rank for details and optional arguments to be passed with xfm.opts. • c(opt1, opt2, etc.)apply the transform specified in opt1, followed by opt2, etc.
xfm.opts	optional arguments to pass to the xfm option specified. Should be a numbered list of lists, where xfm.opts[[i]] corresponds to the optional arguments for xfm[i]. Defaults to the default options for each transform scheme.
robust	whether to perform PCA on a robust estimate of the covariance matrix or not. Defaults to FALSE.
...	trailing args.

Value

A list containing the following:

A	[d,r] the projection matrix from d to r dimensions.
d	the eigen values associated with the eigendecomposition.
Xr	[n,r] the n data points in reduced dimensionality r.

Details

For more details see the help vignette: vignette("pca",package = "lolR")

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.pca(X=X, r=2)  # use pca to project into 2 dimensions
```

`lol.project.pls` *Partial Least-Squares (PLS)*

Description

A function for implementing the Partial Least-Squares (PLS) Algorithm.

Usage

```
lol.project.pls(X, Y, r, ...)
```

Arguments

`X`	[n, d] the data with `n` samples in `d` dimensions.
`Y`	[n] the labels of the samples with `K` unique labels.
`r`	the rank of the projection.
`...`	trailing args.

Value

A list containing the following:

`A`	`[d,r]` the projection matrix from `d` to `r` dimensions.
`ylabs`	`[K]` vector containing the `K` unique, ordered class labels.
`centroids`	`[K,d]` centroid matrix of the `K` unique, ordered classes in native `d` dimensions.
`priors`	`[K]` vector containing the `K` prior probabilities for the unique, ordered classes.
`Xr`	`[n,r]` the `n` data points in reduced dimensionality `r`.
`cr`	`[K,r]` the `K` centroids in reduced dimensionality `r`.

Details

For more details see the help vignette: `vignette("pls",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.pls(X=X, Y=Y, r=5)  # use pls to project into 5 dimensions
```

`lol.project.rp` *Random Projections (RP)*

Description

A function for implementing gaussian random projections (rp).

Usage

```
lol.project.rp(X, r, scale = TRUE, ...)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`r`	the rank of the projection. Note that `r >= K`, and `r < d`.
`scale`	whether to scale the random projection by the sqrt(1/d). Defaults to `TRUE`.
`...`	trailing args.

Value

A list containing the following:

`A`	`[d,r]` the projection matrix from `d` to `r` dimensions.
`Xr`	`[n,r]` the `n` data points in reduced dimensionality `r`.

Details

For more details see the help vignette: `vignette("rp",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.project.rp(X=X, r=5)  # use lol to project into 5 dimensions
```

`lol.sims.cigar` *Stacked Cigar*

Description

A simulation for the stacked cigar experiment.

Usage

```
lol.sims.cigar(n, d, rotate = FALSE, priors = NULL, a = 0.15, b = 4)
```

Arguments

`n`	the number of samples of the simulated data.
`d`	the dimensionality of the simulated data.
`rotate`	whether to apply a random rotation to the mean and covariance. With random rotataion matrix `Q`, `mu = Q*mu`, and `S = Q*S*Q`. Defaults to `FALSE`.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`a`	scalar for all of the mu1 but 2nd dimension. Defaults to `0.15`.
`b`	scalar for 2nd dimension value of mu2 and the 2nd variance term of S. Defaults to `4`.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.cigar(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

`lol.sims.cross` *Cross*

Description

A simulation for the cross experiment, in which the two classes have orthogonal covariant dimensions and the same means.

Usage

```
lol.sims.cross(n, d, rotate = FALSE, priors = NULL, a = 1, b = 0.25, K = 2)
```

Arguments

`n`	the number of samples of simulated data.
`d`	the dimensionality of the simulated data.
`rotate`	With random rotataion matrix `Q`, `mu = Q*mu`, and `S = Q*S*Q`. Defaults to `FALSE`.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`a`	scalar for the magnitude of the variance that is high within the particular class. Defaults to `1`.
`b`	scalar for the magnitude of the varaince that is not high within the particular class. Defaults to `2`.
`K`	the number of classes. Defaults to `2`.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.cross(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

---

lol.sims.fat_tails *Fat Tails Simulation*

Description

A function for simulating from 2 classes with differing means each with 2 sub-clusters, where one sub-cluster has a narrow tail and the other sub-cluster has a fat tail.

Usage

```
lol.sims.fat_tails(
  n,
  d,
  rotate = FALSE,
  f = 15,
  s0 = 10,
  rho = 0.2,
  t = 0.8,
  priors = NULL
)
```

Arguments

n	the number of samples of the simulated data.
d	the dimensionality of the simulated data.
rotate	whether to apply a random rotation to the mean and covariance. With random rotataion matrix Q, mu = Q*mu, and S = Q*S*Q. Defaults to FALSE.
f	the fatness scaling of the tail. S2 = f*S1, where S1_ij = rho if i != j, and 1 if i == j. Defaults to 15.
s0	the number of dimensions with a difference in the means. s0 should be < d. Defaults to 10.
rho	the scaling of the off-diagonal covariance terms, should be < 1. Defaults to 0.2.
t	the fraction of each class from the narrower-tailed distribution. Defaults to 0.8.
priors	the priors for each class. If NULL, class priors are all equal. If not null, should be |priors| = K, a length K vector for K classes. Defaults to NULL.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.fat_tails(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

---

lol.sims.mean_diff *Mean Difference Simulation*

Description

A function for simulating data in which a difference in the means is present only in a subset of dimensions, and equal covariance.

Usage

```
lol.sims.mean_diff(
  n,
  d,
  rotate = FALSE,
  priors = NULL,
  K = 2,
  md = 1,
  subset = c(1),
  offdiag = 0,
  s = 1
)
```

Arguments

n	the number of samples of the simulated data.
d	the dimensionality of the simulated data.
rotate	whether to apply a random rotation to the mean and covariance. With random rotataion matrix Q, mu = Q*mu, and S = Q*S*Q. Defaults to FALSE.
priors	the priors for each class. If NULL, class priors are all equal. If not null, should be |priors| = K, a length K vector for K classes. Defaults to NULL.
K	the number of classes. Defaults to 2.
md	the magnitude of the difference in the means in the specified subset of dimensions. Ddefaults to 1.
subset	the dimensions to have a difference in the means. Defaults to only the first dimension. max(subset) < d. Defaults to c(1).
offdiag	the off-diagonal elements of the covariance matrix. Should be < 1. S_{ij} = offdiag if i != j, or 1 if i == j. Defaults to 0.
s	the scaling parameter of the covariance matrix. S_ij = scaling*1 if i == j, or scaling*offdiag if i != j. Defaults to 1.

Value

A list of class simulation with the following:

X	[n,d] the n data points in d dimensions as a matrix.
Y	[n] the n labels as an array.
mus	[d,K] the K class means in d dimensions.
Sigmas	[d,d,K] the K class covariance matrices in d dimensions.
priors	[K] the priors for each of the K classes.
simtype	The name of the simulation.
params	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: vignette("sims",package = "lolR")

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.mean_diff(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

`lol.sims.qdtoep` *Quadratic Discriminant Toeplitz Simulation*

Description

A function for simulating data generalizing the Toeplitz setting, where each class has a different covariance matrix. This results in a Quadratic Discriminant.

Usage

```
lol.sims.qdtoep(
  n,
  d,
  rotate = FALSE,
  priors = NULL,
  D1 = 10,
  b = 0.4,
  rho = 0.5
)
```

Arguments

`n`	the number of samples of the simulated data.
`d`	the dimensionality of the simulated data.
`rotate`	whether to apply a random rotation to the mean and covariance. With random rotataion matrix `Q`, `mu = Q*mu`, and `S = Q*S*Q`. Defaults to `FALSE`.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`D1`	the dimensionality for the non-equal covariance terms. Defaults to `10`.
`b`	a scaling parameter for the means. Defaults to `0.4`.
`rho`	the scaling of the covariance terms, should be < 1. Defaults to `0.5`.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.qdtoep(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

---

lol.sims.random_rotate

Random Rotation

Description

A helper function for applying a random rotation to gaussian parameter set.

Usage

```
lol.sims.random_rotate(mus, Sigmas, Q = NULL)
```

Arguments

`mus`	means per class.
`Sigmas`	covariances per class.
`Q`	rotation to use, if any

Author(s)

Eric Bridgeford

lol.sims.rev_rtrunk *Reverse Random Trunk*

Description

A simulation for the reversed random trunk experiment, in which the maximal covariant directions are the same as the directions with the maximal mean difference.

Usage

```
lol.sims.rev_rtrunk(
  n,
  d,
  robust = FALSE,
  rotate = FALSE,
  priors = NULL,
  b = 4,
  K = 2,
  maxvar = b^3,
  maxvar.outlier = maxvar^3
)
```

Arguments

n	the number of samples of the simulated data.
d	the dimensionality of the simulated data.
robust	the number of outlier points to add, where outliers have opposite covariance of inliers. Defaults to FALSE, which will not add any outliers.
rotate	whether to apply a random rotation to the mean and covariance. With random rotataion matrix Q, mu = Q*mu, and S = Q*S*Q. Defaults to FALSE.
priors	the priors for each class. If NULL, class priors are all equal. If not null, should be |priors| = K, a length K vector for K classes. Defaults to NULL.
b	scalar for mu scaling. Default to 4.
K	number of classes, should be <4. Defaults to 2.
maxvar	the maximum covariance between the two classes. Defaults to 100.
maxvar.outlier	the maximum covariance for the outlier points. Defaults to maxvar*5.

Value

A list of class simulation with the following:

X	[n,d] the n data points in d dimensions as a matrix.
Y	[n] the n labels as an array.
mus	[d,K] the K class means in d dimensions.

Sigmas	[d,d,K] the K class covariance matrices in d dimensions.
priors	[K] the priors for each of the K classes.
simtype	The name of the simulation.
params	Any extraneous parameters the simulation was created with.
robust	If robust is not false, a list containing inlier a boolean array indicating which points are inliers, s.outlier the covariance structure of outliers, and mu.outlier the means of the outliers.

Details

For more details see the help vignette: vignette("sims",package = "lolR")

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

---

lol.sims.rotation *Sample Random Rotation*

Description

A helper function for estimating a random rotation matrix.

Usage

```
lol.sims.rotation(d)
```

Arguments

d	dimensions to generate a rotation matrix for.

Value

the rotation matrix

Author(s)

Eric Bridgeford

`lol.sims.rtrunk` *Random Trunk*

Description

A simulation for the random trunk experiment, in which the maximal covariant dimensions are the reverse of the maximal mean differences.

Usage

```
lol.sims.rtrunk(
  n,
  d,
  rotate = FALSE,
  priors = NULL,
  b = 4,
  K = 2,
  maxvar = 100
)
```

Arguments

`n`	the number of samples of the simulated data.
`d`	the dimensionality of the simulated data.
`rotate`	whether to apply a random rotation to the mean and covariance. With random rotataion matrix `Q`, `mu = Q*mu`, and `S = Q*S*Q`. Defaults to `FALSE`.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`b`	scalar for mu scaling. Default to `4`.
`K`	number of classes, should be <4. Defaults to `2`.
`maxvar`	the maximum covariance between the two classes. Defaults to `100`.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.
`robust`	If robust is not false, a list containing `inlier` a boolean array indicating which points are inliers, `s.outlier` the covariance structure of outliers, and `mu.outlier` the means of the outliers.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

`lol.sims.sim_gmm` *GMM Simulate*

Description

A helper function for simulating from Gaussian Mixture.

Usage

```
lol.sims.sim_gmm(mus, Sigmas, n, priors)
```

Arguments

`mus`	`[d,K]` the mus for each class.
`Sigmas`	`[d,d,K]` the Sigmas for each class.
`n`	the number of examples.
`priors`	`K` the priors for each class.

Value

A list with the following:

`X`	`[n,d]` the simulated data.
`Y`	`[n]` the labels for each data point.
`priors`	`[K]` the priors for each class.

Author(s)

Eric Bridgeford

`lol.sims.toep` *Toeplitz Simulation*

Description

A function for simulating data in which the covariance is a non-symmetric toeplitz matrix.

Usage

```
lol.sims.toep(n, d, rotate = FALSE, priors = NULL, D1 = 10, b = 0.4, rho = 0.5)
```

Arguments

`n`	the number of samples of the simulated data.
`d`	the dimensionality of the simulated data.
`rotate`	whether to apply a random rotation to the mean and covariance. With random rotataion matrix `Q`, `mu = Q*mu`, and `S = Q*S*Q`. Defaults to `FALSE`.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`D1`	the dimensionality for the non-equal covariance terms. Defaults to `10`.
`b`	a scaling parameter for the means. Defaults to `0.4`.
`rho`	the scaling of the covariance terms, should be < 1. Defaults to `0.5`/

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.toep(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

`lol.sims.xor2` *Xor Problem*

Description

A function to simulate from the 2-class xor problem.

Usage

```
lol.sims.xor2(n, d, priors = NULL, fall = 100)
```

Arguments

`n`	the number of samples of the simulated data.
`d`	the dimensionality of the simulated data.
`priors`	the priors for each class. If `NULL`, class priors are all equal. If not null, should be `|priors| = K`, a length `K` vector for `K` classes. Defaults to `NULL`.
`fall`	the falloff for the covariance structuring. Sigma declines by ndim/fall across the variance terms. Defaults to `100`.

Value

A list of class `simulation` with the following:

`X`	`[n,d]` the `n` data points in `d` dimensions as a matrix.
`Y`	`[n]` the `n` labels as an array.
`mus`	`[d,K]` the `K` class means in `d` dimensions.
`Sigmas`	`[d,d,K]` the `K` class covariance matrices in `d` dimensions.
`priors`	`[K]` the priors for each of the `K` classes.
`simtype`	The name of the simulation.
`params`	Any extraneous parameters the simulation was created with.

Details

For more details see the help vignette: `vignette("sims",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.xor2(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
```

`lol.utils.decomp` *A utility to use irlba when necessary*

Description

A utility to use irlba when necessary

Usage

```
lol.utils.decomp(
  X,
  xfm = FALSE,
  xfm.opts = list(),
  ncomp = 0,
  t = 0.05,
  robust = FALSE
)
```

Arguments

`X`	the data to compute the svd of.
`xfm`	whether to transform the variables before taking the SVD. • FALSEapply no transform to the variables. • 'unit'unit transform the variables, defaulting to centering and scaling to mean 0, variance 1. See `scale` for details and optional args. • 'log'log-transform the variables, for use-cases such as having high variance in larger values. Defaults to natural logarithm. See `log` for details and optional args. • 'rank'rank-transform the variables. Defalts to breaking ties with the average rank of the tied values. See `rank` for details and optional args. • c(opt1, opt2, etc.)apply the transform specified in opt1, followed by opt2, etc.
`xfm.opts`	optional arguments to pass to the `xfm` option specified. Should be a numbered list of lists, where `xfm.opts[[i]]` corresponds to the optional arguments for `xfm[i]`. Defaults to the default options for each transform scheme.
`ncomp`	the number of left singular vectors to retain.
`t`	the threshold of percent of singular vals/vecs to use irlba.
`robust`	whether to use a robust estimate of the covariance matrix when taking PCA. Defaults to `FALSE`.

Value

the svd of X.

Author(s)

Eric Bridgeford

`lol.utils.deltas` *A function that performs a utility computation of information about the differences of the classes.*

Description

A function that performs a utility computation of information about the differences of the classes.

Usage

```
lol.utils.deltas(centroids, priors, ...)
```

Arguments

`centroids`	`[d,K]` centroid matrix of the unique, ordered classes.
`priors`	`[K]` vector containing prior probability for the unique, ordered classes.
`...`	optional args.

Value

deltas `[d,K]` the K difference vectors.

Author(s)

Eric Bridgeford

`lol.utils.info` *A function that performs basic utilities about the data.*

Description

A function that performs basic utilities about the data.

Usage

```
lol.utils.info(X, Y, robust = FALSE, ...)
```

Arguments

X	[n,d] the data with n samples in d dimensions.
Y	[n] the labels of the samples.
robust	whether to perform PCA on a robust estimate of the covariance matrix or not. Defaults to FALSE.
...	optional args.

Value

n the number of samples.

d the number of dimensions.

ylabs [K] vector containing the unique, ordered class labels.

priors [K] vector containing prior probability for the unique, ordered classes.

Author(s)

Eric Bridgeford

---

lol.utils.ohe *A function for one-hot encoding categorical respose vectors.*

Description

A function for one-hot encoding categorical respose vectors.

Usage

```
lol.utils.ohe(Y)
```

Arguments

Y	[n] a vector of the categorical resposes, with K unique categories.

Value

a list containing the following:

Yh	[n, K] the one-hot encoded Y respose variable.
ylabs	[K] a vector of the y names corresponding to each response column.

Author(s)

Eric Bridgeford

`lol.xval.eval` *Embedding Cross Validation*

Description

A function for performing leave-one-out cross-validation for a given embedding model. This function produces fold-wise cross-validated misclassification rates for standard embedding techniques. Users can optionally specify custom embedding techniques with proper configuration of `alg.*` parameters and hyperparameters. Optional classifiers implementing the S3 `predict` function can be used for classification, with hyperparameters to classifiers for determining misclassification rate specified in `classifier.*` parameters and hyperparameters.

Usage

```
lol.xval.eval(
  X,
  Y,
  r,
  alg,
  sets = NULL,
  alg.dimname = "r",
  alg.opts = list(),
  alg.embedding = "A",
  classifier = lda,
  classifier.opts = list(),
  classifier.return = "class",
  k = "loo",
  rank.low = FALSE,
  ...
)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels.
`r`	the number of embedding dimensions desired, where `r` <= `d`.
`alg`	the algorithm to use for embedding. Should be a function that accepts inputs `X`, `Y`, and has a parameter for `alg.dimname` if `alg` is supervised, or just `X` and `alg.dimname` if `alg` is unsupervised.This algorithm should return a list containing a matrix that embeds from d to r <= d dimensions.
`sets`	a user-defined cross-validation set. Defaults to `NULL`.

- `is.null(sets)` randomly partition the inputs `X` and `Y` into training and testing sets.

- `!is.null(sets)` use a user-defined partitioning of the inputs `X` and `Y` into training and testing sets. Should be in the format of the outputs from `lol.xval.split`. That is, a `list` with each element containing `X.train`, an `[n-k][d]` subset of data to test on, `Y.train`, an `[n-k]` subset of class labels for `X.train`; `X.test`, an `[n-k][d]` subset of data to test the model on, `Y.train`, an `[k]` subset of class labels for `X.test`.

`alg.dimname` the name of the parameter accepted by `alg` for indicating the embedding dimensionality desired. Defaults to `r`.

`alg.opts` the hyper-parameter options you want to pass into your algorithm, as a keyworded list. Defaults to `list()`, or no hyper-parameters.

`alg.embedding` the attribute returned by `alg` containing the embedding matrix. Defaults to assuming that `alg` returns an embgedding matrix as `"A"`.

- `!is.nan(alg.embedding)` Assumes that `alg` will return a list containing an attribute, `alg.embedding`, a `[d,r]` matrix that embeds `[n,d]` data from `[d]` to `[r < d]` dimensions.
- `is.nan(alg.embedding)` Assumes that `alg` returns a `[d,r]` matrix that embeds `[n,d]` data from `[d]` to `[r < d]` dimensions.

`classifier` the classifier to use for assessing performance. The classifier should accept `X`, a `[n,d]` array as the first input, and `Y`, a `[n]` array of labels, as the first 2 arguments. The class should implement a predict function, `predict.classifier`, that is compatible with the `stats::predict` `S3` method. Defaults to `MASS::lda`.

`classifier.opts`
any extraneous options to be passed to the classifier function, as a list. Defaults to an empty list.

`classifier.return`
if the return type is a list, `class` encodes the attribute containing the prediction labels from `stats::predict`. Defaults to the return type of `MASS::lda`, `class`.

- `!is.nan(classifier.return)` Assumes that `predict.classifier` will return a list containing an attribute, `classifier.return`, that encodes the predicted labels.
- `is.nan(classifier.return)` Assumes that `predict.classifer` returns a `[n]` vector/array containing the prediction labels for `[n,d]` inputs.

`k` the cross-validated method to perform. Defaults to `'loo'`. If `sets` is provided, this option is ignored. See `lol.xval.split` for details.

- `'loo'` Leave-one-out cross validation
- `isinteger(k)` perform `k`-fold cross-validation with `k` as the number of folds.

`rank.low` whether to force the training set to low-rank. Defaults to `FALSE`. If `sets` is provided, this option is ignored. See `lol.xval.split` for details.

- if `rank.low == FALSE`, uses default cross-validation method with standard `k`-fold validation. Training sets are `k-1` folds, and testing sets are `1` fold, where the fold held-out for testing is rotated to ensure no dependence of potential downstream inference in the cross-validated misclassification rates.

- if]coderank.low == TRUE, users cross-validation method with `ntrain = min((k-1)/k*n,d)` sample training sets, where `d` is the number of dimensions in `X`. This ensures that the training data is always low-rank, `ntrain < d + 1`. Note that the resulting training sets may have `ntrain < (k-1)/k*n`, but the resulting testing sets will always be properly rotated `ntest = n/k` to ensure no dependencies in fold-wise testing.

`...` trailing args.

Value

Returns a list containing:

`lhat`	the mean cross-validated error.
`model`	The model returned by `alg` computed on all of the data.
`classifier`	The classifier trained on all of the embedded data.
`lhats`	the cross-validated error for each of the `k`-folds.

Details

For more details see the help vignette: `vignette("xval",package = "lolR")`

For extending cross-validation techniques shown here to arbitrary embedding algorithms, see the vignette: `vignette("extend_embedding",package = "lolR")`

For extending cross-validation techniques shown here to arbitrary classification algorithms, see the vignette: `vignette("extend_classification",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
# train model and analyze with loo validation using lda classifier
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
r=5  # embed into r=5 dimensions
# run cross-validation with the nearestCentroid method and
# leave-one-out cross-validation, which returns only
# prediction labels so we specify classifier.return as NaN
xval.fit <- lol.xval.eval(X, Y, r, lol.project.lol,
                          classifier=lol.classify.nearestCentroid,
                          classifier.return=NaN, k='loo')

# train model and analyze with 5-fold validation using lda classifier
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
xval.fit <- lol.xval.eval(X, Y, r, lol.project.lol, k=5)

# pass in existing cross-validation sets
```

```
sets <- lol.xval.split(X, Y, k=2)
xval.fit <- lol.xval.eval(X, Y, r, lol.project.lol, sets=sets)
```

`lol.xval.optimal_dimselect` *Optimal Cross-Validated Number of Embedding Dimensions*

Description

A function for performing leave-one-out cross-validation for a given embedding model, that allows users to determine the optimal number of embedding dimensions for their algorithm-of-choice. This function produces fold-wise cross-validated misclassification rates for standard embedding techniques across a specified selection of embedding dimensions. Optimal embedding dimension is selected as the dimension with the lowest average misclassification rate across all folds. Users can optionally specify custom embedding techniques with proper configuration of `alg.*` parameters and hyperparameters. Optional classifiers implementing the S3 `predict` function can be used for classification, with hyperparameters to classifiers for determining misclassification rate specified in `classifier.*`.

Usage

```
lol.xval.optimal_dimselect(
  X,
  Y,
  rs,
  alg,
  sets = NULL,
  alg.dimname = "r",
  alg.opts = list(),
  alg.embedding = "A",
  alg.structured = TRUE,
  classifier = lda,
  classifier.opts = list(),
  classifier.return = "class",
  k = "loo",
  rank.low = FALSE,
  ...
)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels. Defaults to `NaN`.#' @param alg.opts any extraneous options to be passed to the classifier function, as a list. Defaults to an empty list. For example, this could be the embedding dimensionality to investigate.
`rs`	`[r,n]` the embedding dimensions to investigate over, where `max(rs)` <= `d`.

`alg` the algorithm to use for embedding. Should be a function that accepts inputs `X` and `Y` and embedding dimension `r` if `alg` is supervised, or just `X` and embedding dimension `r` if `alg` is unsupervised.This algorithm should return a list containing a matrix that embeds from d to r < d dimensions.

`sets` a user-defined cross-validation set. Defaults to `NULL`.

- `is.null(sets)` randomly partition the inputs `X` and `Y` into training and testing sets.
- `!is.null(sets)` use a user-defined partitioning of the inputs `X` and `Y` into training and testing sets. Should be in the format of the outputs from `lol.xval.split`. That is, a `list` with each element containing `X.train`, an `[n-k][d]` subset of data to test on, `Y.train`, an `[n-k]` subset of class labels for `X.train`; `X.test`, an `[n-k][d]` subset of data to test the model on, `Y.train`, an `[k]` subset of class labels for `X.test`.

`alg.dimname` the name of the parameter accepted by `alg` for indicating the embedding dimensionality desired. Defaults to `r`.

`alg.opts` the hyper-parameter options to pass to your algorithm as a keyworded list. Defaults to `list()`, or no hyper-parameters. This should not include the number of embedding dimensions, `r`, which are passed separately in the `rs` vector.

`alg.embedding` the attribute returned by `alg` containing the embedding matrix. Defaults to assuming that `alg` returns an embgedding matrix as `"A"`.

- `!is.nan(alg.embedding)` Assumes that `alg` will return a list containing an attribute, `alg.embedding`, a `[d,r]` matrix that embeds `[n,d]` data from `[d]` to `[r < d]` dimensions.
- `is.nan(alg.embedding)` Assumes that `alg` returns a `[d,r]` matrix that embeds `[n,d]` data from `[d]` to `[r < d]` dimensions.

`alg.structured` a boolean to indicate whether the embedding matrix is structured. Provides performance increase by not having to compute the embedding matrix `xv` times if unnecessary. Defaults to `TRUE`.

- `TRUE` assumes that if `Ar: R^d -> R^r` embeds from `d` to `r` dimensions and `Aq: R^d -> R^q` from `d` to `q > r` dimensions, that `Aq[,1:r] == Ar`,
- `TRUE` assumes that if `Ar: R^d -> R^r` embeds from `d` to `r` dimensions and `Aq: R^d -> R^q` from `d` to `q > r` dimensions, that `Aq[,1:r] != Ar`,

`classifier` the classifier to use for assessing performance. The classifier should accept `X`, a `[n,d]` array as the first input, and `Y`, a `[n]` array of labels, as the first 2 arguments. The class should implement a predict function, `predict.classifier`, that is compatible with the `stats::predict` `S3` method. Defaults to `MASS::lda`.

`classifier.opts` any extraneous options to be passed to the classifier function, as a list. Defaults to an empty list.

`classifier.return` if the return type is a list, `class` encodes the attribute containing the prediction labels from `stats::predict`. Defaults to the return type of `MASS::lda`, `class`.

- `!is.nan(classifier.return)` Assumes that `predict.classifier` will return a list containing an attribute, `classifier.return`, that encodes the predicted labels.

- `is.nan(classifier.return)` Assumes that `predict.classifer` returns a `[n]` vector/array containing the prediction labels for `[n,d]` inputs.

`k` the cross-validated method to perform. Defaults to `'loo'`. If `sets` is provided, this option is ignored. See `lol.xval.split` for details.

- `'loo'` Leave-one-out cross validation
- `isinteger(k)` perform `k`-fold cross-validation with `k` as the number of folds.

`rank.low` whether to force the training set to low-rank. Defaults to `FALSE`. If `sets` is provided, this option is ignored. See `lol.xval.split` for details.

- if `rank.low == FALSE`, uses default cross-validation method with standard `k`-fold validation. Training sets are `k-1` folds, and testing sets are `1` fold, where the fold held-out for testing is rotated to ensure no dependence of potential downstream inference in the cross-validated misclassification rates.
- if]coderank.low == TRUE, users cross-validation method with `ntrain = min((k-1)/k*n,d)` sample training sets, where `d` is the number of dimensions in `X`. This ensures that the training data is always low-rank, `ntrain < d + 1`. Note that the resulting training sets may have `ntrain < (k-1)/k*n`, but the resulting testing sets will always be properly rotated `ntest = n/k` to ensure no dependencies in fold-wise testing.

`...` trailing args.

Value

Returns a list containing:

`folds.data` the results, as a data-frame, of the per-fold classification accuracy.

`foldmeans.data` the results, as a data-frame, of the average classification accuracy for each `r`.

`optimal.lhat` the classification error of the optimal `r`

.

`optimal.r` the optimal number of embedding dimensions from `rs`

.

`model` the model trained on all of the data at the optimal number of embedding dimensions.

`classifier` the classifier trained on all of the data at the optimal number of embedding dimensions.

Details

For more details see the help vignette: `vignette("xval",package = "lolR")`

For extending cross-validation techniques shown here to arbitrary embedding algorithms, see the vignette: `vignette("extend_embedding",package = "lolR")`

For extending cross-validation techniques shown here to arbitrary classification algorithms, see the vignette: `vignette("extend_classification",package = "lolR")`

Author(s)

Eric Bridgeford

Examples

```
# train model and analyze with loo validation using lda classifier
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
# run cross-validation with the nearestCentroid method and
# leave-one-out cross-validation, which returns only
# prediction labels so we specify classifier.return as NaN
xval.fit <- lol.xval.optimal_dimselect(X, Y, rs=c(5, 10, 15), lol.project.lol,
                          classifier=lol.classify.nearestCentroid,
                          classifier.return=NaN, k='loo')

# train model and analyze with 5-fold validation using lda classifier
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
xval.fit <- lol.xval.optimal_dimselect(X, Y, rs=c(5, 10, 15), lol.project.lol, k=5)

# pass in existing cross-validation sets
sets <- lol.xval.split(X, Y, k=2)
xval.fit <- lol.xval.optimal_dimselect(X, Y, rs=c(5, 10, 15), lol.project.lol, sets=sets)
```

---

lol.xval.split *Cross-Validation Data Splitter*

Description

A function to split a dataset into training and testing sets for cross validation. The procedure for cross-validation is to split the data into k-folds. The k-folds are then rotated individually to form a single held-out testing set the model will be validated on, and the remaining (k-1) folds are used for training the developed model. Note that this cross-validation function includes functionality to be used for low-rank cross-validation. In that case, instead of using the full (k-1) folds for training, we subset `min((k-1)/k*n,d)` samples to ensure that the resulting training sets are all low-rank. We still rotate properly over the held-out fold to ensure that the resulting testing sets do not have any shared examples, which would add a complicated dependence structure to inference we attempt to infer on the testing sets.

Usage

```
lol.xval.split(X, Y, k = "loo", rank.low = FALSE, ...)
```

Arguments

`X`	`[n,d]` the data with `n` samples in `d` dimensions.
`Y`	`[n]` the labels of the samples with `K` unique labels.

k — the cross-validated method to perform. Defaults to 'loo'.

- if k == round(k), performed k-fold cross-validation.
- if k == 'loo', performs leave-one-out cross-validation.

rank.low — whether to force the training set to low-rank. Defaults to FALSE.

- if rank == FALSE, uses default cross-validation method with standard k-fold validation. Training sets are k-1 folds, and testing sets are 1 fold, where the fold held-out for testing is rotated to ensure no dependence of potential downstream inference in the cross-validated misclassification rates.
- if rank == TRUE, users cross-validation method with ntrain = min((k-1)/k*n,d) sample training sets, where d is the number of dimensions in X. This ensures that the training data is always low-rank, ntrain < d + 1. Note that the resulting training sets may have ntrain < (k-1)/k*n, but the resulting testing sets will always be properly rotated ntest = n/k to ensure no dependencies in fold-wise testing.

... — optional args.

Value

sets the cross-validation sets as an object of class "XV" containing the following:

train — length [ntrain] vector indicating the indices of the training examples.

test — length [ntest] vector indicating the indices of the testing examples.

Author(s)

Eric Bridgeford

Examples

```
# prepare data for 10-fold validation
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
sets.xval.10fold <- lol.xval.split(X, Y, k=10)

# prepare data for loo validation
sets.xval.loo <- lol.xval.split(X, Y, k='loo')
```

---

predict.nearestCentroid

Nearest Centroid Classifier Prediction

Description

A function that predicts the class of points based on the nearest centroid

Usage

```
## S3 method for class 'nearestCentroid'
predict(object, X, ...)
```

Arguments

object	An object of class nearestCentroid, with the following attributes: • centroids[K,d] the centroids of each class with K classes in d dimensions. • ylabs[K] the ylabels for each of the K unique classes, ordered. • priors[K] the priors for each of the K classes.
X	[n,d] the data to classify with n samples in d dimensions.
...	optional args.

Value

Yhat [n] the predicted class of each of the n data point in X.

Details

For more details see the help vignette: vignette("centroid",package = "lolR")

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.nearestCentroid(X, Y)
Yh <- predict(model, X)
```

---

predict.randomChance *Randomly Chance Classifier Prediction*

Description

A function that predicts the maximally present class in the dataset. Functionality consistent with the standard R prediction interface so that one can compute the "chance" accuracy with minimal modification of other classification scripts.

Usage

```
## S3 method for class 'randomChance'
predict(object, X, ...)
```

Arguments

object	An object of class randomChance, with the following attributes: • ylabs[K] the ylabels for each of the K unique classes, ordered. • priors[K] the priors for each of the K classes.
X	[n,d] the data to classify with n samples in d dimensions.
...	optional args.

Value

Yhat [n] the predicted class of each of the n data point in X.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.randomChance(X, Y)
Yh <- predict(model, X)
```

---

predict.randomGuess *Randomly Guessing Classifier Prediction*

Description

A function that predicts by randomly guessing based on the pmf of the class priors. Functionality consistent with the standard R prediction interface so that one can compute the "guess" accuracy with minimal modification of other classification scripts.

Usage

```
## S3 method for class 'randomGuess'
predict(object, X, ...)
```

Arguments

object	An object of class randomGuess, with the following attributes: • ylabs[K] the ylabels for each of the K unique classes, ordered. • priors[K] the priors for each of the K classes.
X	[n,d] the data to classify with n samples in d dimensions.
...	optional args.

Value

Yhat [n] the predicted class of each of the n data point in X.

Author(s)

Eric Bridgeford

Examples

```
library(lolR)
data <- lol.sims.rtrunk(n=200, d=30)  # 200 examples of 30 dimensions
X <- data$X; Y <- data$Y
model <- lol.classify.randomGuess(X, Y)
Yh <- predict(model, X)
```

Index