Package ‘iLDA’

December 8, 2018

Type Package
Title Integrative Linear Discriminant Analysis
Version 0.1.0
Author Quefeng Li
Maintainer Quefeng Li <quefeng@email.NuncNedu>

Description
An integrative linear discriminant analysis classifier that integrates multimodal data for binary classification. Its applications include but not limit to multi-omics data and multimodal neuroimaging data. The package also allows the data to have blocks of missing values, i.e. one sample can miss the measurements of an entire modality. The package is based on a proximal gradient algorithm. For more details, please see the paper in the reference.

License GPL-3

Imports Matrix, doParallel, foreach, robustbase


RoxygenNote 6.0.1

R topics documented:

classify ................................................................. 1
cv.iLDA ............................................................... 2
iLDA ................................................................. 3

Index 7

classify Classify new samples based on the integrative linear discriminant analysis rule

Description
Classify a new sample to class 0 if and only if \( \hat{\beta}^T \{ x - (\hat{\mu}_0 + \hat{\mu}_1)/2 \} + \log(\hat{\pi}_0/\hat{\pi}_1) \geq 0 \), where \( \hat{\beta} \) is the solution of the iLDA problem, \( x \) is the vector of the new sample, \( \hat{\mu}_0 \) and \( \hat{\mu}_1 \) are the estimators of the means of the two classes, and \( \hat{\pi}_0 \) and \( \hat{\pi}_1 \) are the estimators of the prior probabilities of the two classes.
Usage

```r
classify(obj, new.X, balanced = T)
```

Arguments

- **obj**: an object returned by `iLDA`.
- **new.X**: a matrix of new samples to be classified. Rows are samples and columns are features.
- **balanced**: a logical flag of whether the two classes have equal prior probability. If balanced = TRUE, enforce \( \bar{\pi}_0 = \bar{\pi}_1 = 1/2 \); otherwise, use the estimators returned by `iLDA`.

Value

a vector of predicted class labels of the new samples.

---

### cv.iLDA

**Cross-validation to choose the optimal tuning parameters**

**Description**

Choose the optimal tuning parameters in the integrative linear discriminant analysis (iLDA) problem by the K-fold cross-validation.

**Usage**

```r
cv.iLDA(Y, X, group, alpha.grid, lambda.grid, K = 5, missing = F, robust = F, k = 1.345, tol = 0.001, max.iter = 200, balanced = T)
```

Arguments

- **Y**: a vector of class labels.
- **X**: a matrix of samples. The dimension is \( nobs \times nvars \), where \( nobs \) is the sample size in the training set and \( nvars \) is number of features in all modalities. In practice, cbind data from all modalities to render \( X \).
- **group**: a list of group indices. Each element in the list is the column indices of variables belonging to that group in \( X \). See an example in Examples of `iLDA`.
- **alpha.grid**: the search grip of \( \alpha \).
- **lambda.grid**: the search grip of \( \lambda \).
- **K**: number of folds in cross-validation.
- **missing**: a logical flag of whether \( X \) contains missing values.
- **robust**: a logical flag of whether using robust estimators for \( \mu_0, \mu_1 \) and \( \Sigma \). If TRUE, the Huber robust estimators will be used. For more details, please see the reference.
- **k**: robustification factor in the Huber estimator.
- **tol**: tolerance level for stopping the algorithm.
- **max.iter**: maximum number of iterations allowed.
- **balanced**: see definition in `classify`.
iLDA

Value

- best.alpha: best value of α.
- best.lambda: best value of λ.
- cv.error: a matrix of cross-validated misclassification rates.

Description

Solve the integrative linear discriminant analysis (iLDA) problem via a proximal gradient algorithm.

Usage

iLDA(y, X, group, alpha, lambda, missing = F, robust = F, k = 1.345, 
      tol = 0.001, max.iter = 200)

Arguments

- Y: a vector of class labels.
- X: a matrix of samples. The dimension is nobs * nvars, where nobs is the sample size in the training set and nvars is number of features in all modalities. In practice, cbind data from all modalities to render X.
- group: a list of group indices. Each element in the list is the column indices of variables belonging to that group in X. See an example in Examples of iLDA.
- alpha: the tuning parameter α.
- lambda: the tuning parameter λ.
- missing: a logical flag of whether X contains missing values.
- robust: a logical flag of whether using robust estimators for μ0, μ1 and Σ. If TRUE, the Huber robust estimators will be used. For more details, please see the reference.
- k: robustification factor in the Huber estimator.
- tol: tolerance level for stopping the algorithm.
- max.iter: maximum number of iterations allowed.

Details

The function solves a regularized minimization problem

\[ \hat{\beta} = \arg\min_{\beta} \frac{1}{2} \beta^T \hat{\Sigma} \beta - (\hat{\mu}_0 - \hat{\mu}_1)^T \beta + \lambda \left( \sum_{j \in N} \| \beta_{S_j} \|_1 + \sum_{j \in M} \| \beta_{S_j} \|_G \right), \]

where \( \hat{\Sigma} \) is the pooled estimator of covariance of the two classes, \( \hat{\mu}_0 \) and \( \hat{\mu}_1 \) are the estimators of the means of the two classes. \( N \) is the set of variables appearing in only one modality. An \( L_1 \) penalty is imposed on these variables. \( M \) is the set of variables appearing in multiple modalities. A sparse group Lasso penalty imposed on these variables. Such a penalty is defined as

\[ \| \beta_{S_j} \|_G = (1 - \alpha) \| \beta_{S_j} \|_1 + \alpha \| \beta_{S_j} \|_2, \quad \alpha \in [0, 1]. \]
Value

a list with elements

beta       solution of the iLDA problem.
mu0        estimator of $\mu_0$.
mu1        estimator of $\mu_1$.
mu         estimator of $(\mu_0 + \mu_1)/2$.
Sigma.hat  estimator of $\Sigma$.
w0         estimator of log odds of the prior probabilities of the two classes, i.e. $\log(\hat{\pi}_0/\hat{\pi}_1)$.
iter.count actual number of iterations.
converge   a logical flag of whether the algorithm converges.

Examples

```r
## not run:
## A toy simulation
library(MASS)

n <- 50                      # sample size
p <- 100                     # number of variables in each modality
M <- 3                       # number of modalities
percentage <- 0.2
share <- p * percentage

Sigma <- matrix(1, M * p, M * p)          # true covariance matrix
for (i in 1:(M * p)) {
  for (j in 1:(M * p)) {
    Sigma[i, j] <- 0.8^abs(i - j)
  }
}

beta.1 <- c(rep(c(rep(0.3, 2), rep(0, share - 2)), M),
             rep(0.3, 4), rep(0, 2 * share - 4))

beta <- rep(beta.1, M)            # true beta

delta <- Sigma %*% beta


group <- vector('list', 2 * p)    # generate group indices
# modality 1
for (i in 1:(M * share)) {
  group[[i]] <- i
}
for (i in (4 * share + 1):(6 * share)) {
  group[[i]] <- i - share
}
# modality 2
for (i in 1:(2 * share)) {
  group[[i]] <- c(group[[i]], p + i)
}
for (i in (M * share + 1):(4 * share)) {
  group[[i]] <- p - share + i
}
for (i in (6 * share + 1):(8 * share)) {
  group[[i]] <- p - M * share + i
}
```

# modality M
for (i in 1:share) {
    group[[i]] <- c(group[[i]], 2 * p + i)
}
for (i in (2 * share + 1):(4 * share)) {
    group[[i]] <- c(group[[i]], 2 * p - share + i)
}
for (i in (8 * share + 1):(10 * share)) {
    group[[i]] <- 2 * p - 5 * share + i
}
group.ind <- vector('list', p)
for (i in 1:p) {
    group.ind[[i]] <- i
}

## searching grid of (alpha, lambda)
alpha <- seq(0, 1, len = 2)
lambda <- 2^seq(-4, 1, len = 3)

## generate training set
X.trn <- rbind(mvrnorm(n, mu = rep(0, M * p), Sigma = Sigma),
               mvrnorm(n, mu = -delta, Sigma = Sigma))
Y.trn <- c(rep(0, n), rep(1, n))

## generate test set
X.tst <- rbind(mvrnorm(n, mu = rep(0, M * p), Sigma = Sigma),
               mvrnorm(n, mu = -delta, Sigma = Sigma))
Y.tst <- c(rep(0, n), rep(1, n))

## cv and fit
cv <- cv.lda(Y.trn, X.trn, group, alpha, lambda)
fit <- ilDA(Y.trn, X.trn, group,
            alpha = cv$best.alpha,
            lambda = cv$best.lambda)

## prediction
Y.prd <- classify(fit, X.tst)
(error.rate <- sum((Y.tst - Y.prd)^2) / length(Y.tst))

## generate training set with block missing values
missing.prob <- 0.05
X.trn.missing <- cbind(X.trn[, 1:p] * ifelse(rbinom(n, 1, missing.prob) == 1, NA, 1),
                        X.trn[, (p + 1):(2 * p)] * ifelse(rbinom(n, 1, missing.prob) == 1, NA, 1),
                        X.trn[, (2 * p + 1):(3 * p)] * ifelse(rbinom(n, 1, missing.prob) == 1, NA, 1))

## cv and fit
cv <- cv.lda(Y.trn, X.trn.missing, group, alpha, lambda, missing = T)
fit <- ilDA(Y.trn, X.trn.missing, group,
            alpha = cv$best.alpha,
            lambda = cv$best.lambda,
            missing = T)

## prediction
Y.prd <- classify(fit, X.tst)
(error.rate <- sum((Y.tst - Y.prd)^2) / length(Y.tst))
Index

classify, 1, 2

cv.iLDA, 2

iLDA, 2, 3, 3