Package 'MFKnockoffs'

September 29, 2017

Type Package

Title Model-Free Knockoff Filter for Controlled Variable Selection

Version 0.9.1

Date 2017-09-27

Description Model-free knockoffs are a general procedure for controlling the false discovery rate (FDR) when performing variable selection. For more information, see the website below and the accompanying paper: Candes et al., ``Panning for Gold: Model-free Knockoffs for High-dimensional Controlled Variable Selection'', 2016, <arXiv:1610.02351>.

License GPL-3

URL https://statweb.stanford.edu/~candes/MF_Knockoffs/index.html

Depends methods, stats

Imports Rdsdp, Matrix, corpcor, glmnet, RSpectra, gtools

Suggests knitr, testthat, rmarkdown, lars, ranger, stabs, flare, doMC, parallel

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Rina Foygel Barber [ctb] (Development of the original fixed-design Knockoffs),
Emmanuel Candes [ctb] (Development of Model-Free Knockoffs and original fixed-design Knockoffs),
Lucas Janson [ctb] (Development of Model-Free Knockoffs),
Evan Patterson [aut] (Original R package for the original fixed-design Knockoffs),
Matteo Sesia [aut, cre] (R package for Model-Free Knockoffs)

Maintainer Matteo Sesia <msesia@stanford.edu>

Repository CRAN

Date/Publication 2017-09-29 08:47:26 UTC

R topics documented:

`MFKnockoffs` *Model-free controlled variable selection*

Description

Model-free knockoffs provide a general and powerful tool to perform high-dimensional controlled variable selection.

Details

The method constructs artificial 'knockoff copies' of the variables in a statistical model and then selects those variables that are clearly better than their corresponding fake copies. A wide range of statistics and machine learning tools can be exploited to estimate the importance of each feature, while guaranteeing finite-sample control of the false discovery rate (FDR). This model-free approach makes it possible to use knockoffs for data originating from any conditional model (Y|X), no matter how high-dimensional, provided that the distribution of the covariates (X) is known.

For more information, see the website below and the accompanying paper.

https://statweb.stanford.edu/~candes/MF_Knockoffs/index.html

`MFKnockoffs.create.approximate_gaussian` *Sample approximate second-order multivariate Gaussian knockoff variables*

Description

Samples approximate second-order multivariate Gaussian knockoff variables for the original variables.

Usage

```
MFKnockoffs.create.approximate_gaussian(X, method = c("asdp", "equi", "sdp"),
  shrink = F)
```

Arguments

`X`	normalized n-by-p realization of the design matrix
`method`	either 'equi', 'sdp' or 'asdp' (default:'asdp') This will be computed according to 'method', if not supplied
`shrink`	whether to shrink the estimated covariance matrix (default: FALSE)

Details

If the argument `shrink` is set to TRUE, a James-Stein-type shrinkage estimator for the covariance matrix is used instead of the traditional maximum-likelihood estimate. This option requires the package `corpcor`. Type `?corpcor::cov.shrink` for more details.

Even if the argument `shrink` is set to FALSE, in the case that the estimated covariance matrix is not positive-definite, this function will apply some shrinkage.

To use SDP knockoffs, you must have a Python installation with CVXPY. For more information, see the vignette on SDP knockoffs: `vignette('sdp', package='MFKnockoffs')`

Value

n-by-p matrix of knockoff variables

References

Candes et al., Panning for Gold: Model-free Knockoffs for High-dimensional Controlled Variable Selection, arXiv:1610.02351 (2016). https://statweb.stanford.edu/~candes/MF_Knockoffs/index.html

See Also

Other methods for creating knockoffs: `MFKnockoffs.create.fixed`, `MFKnockoffs.create.gaussian`

`MFKnockoffs.create.fixed` *Create fixed-design knockoff variables*

Description

Creates fixed-design knockoff variables for the original variables.

Usage

```
MFKnockoffs.create.fixed(X, method = c("sdp", "equi"), sigma = NULL,
  y = NULL, randomize = F)
```

Arguments

`X`	normalized n-by-p design matrix (n >= 2p)
`method`	either 'equi' or 'sdp' (default:'sdp')
`sigma`	noise level, used to augment the data with extra rows if necessary (default: NULL)
`y`	vector of observed responses, used to estimate the noise level if 'sigma' is not provided (default: NULL)
`randomize`	whether the knockoffs are deterministic or randomized (default:False)

Value

An object of class "MFKnockoffs.variables". This object is a list containing at least the following components:

`X`	The n-by-p matrix of original variables (possibly augmented or transformed)
`X_k`	The n-by-p matrix of knockoff variables
`y`	The vector of observed responses (possibly augmented)

References

Barber and Candes, Controlling the false discovery rate via knockoffs. Ann. Statist. 43 (2015), no. 5, 2055–2085. https://projecteuclid.org/euclid.aos/1438606853

Fixed-design knockoff assume a linear regression model for Y|X. Moreover, they only guarantee FDR control with statistics satisfying the "sufficiency" property. In particular, the default statistics with cross-validated lasso does not satisfy this property and should not be used with fixed-design knockoffs.

See Also

Other methods for creating knockoffs: `MFKnockoffs.create.approximate_gaussian`, `MFKnockoffs.create.gaussian`

`MFKnockoffs.create.gaussian` *Sample multivariate Gaussian knockoff variables*

Description

Samples multivariate Gaussian fixed-design knockoff variables for the original variables.

Usage

```
MFKnockoffs.create.gaussian(X, mu, Sigma, method = c("asdp", "sdp", "equi"),
  diag_s = NULL)
```

Arguments

`X`	normalized n-by-p realization of the design matrix
`mu`	mean vector of length p for X
`Sigma`	p-by-p covariance matrix for X
`method`	either 'equi', 'sdp' or 'asdp' (default:'asdp')
`diag_s`	pre-computed vector of covariances between the original variables and the knockoffs. This will be computed according to 'method', if not supplied

Value

n-by-p matrix of knockoff variables

References

Candes et al., Panning for Gold: Model-free Knockoffs for High-dimensional Controlled Variable Selection, arXiv:1610.02351 (2016). https://statweb.stanford.edu/~candes/MF_Knockoffs/index.html

See Also

Other methods for creating knockoffs: `MFKnockoffs.create.approximate_gaussian`, `MFKnockoffs.create.fixed`

`MFKnockoffs.filter` *Model-Free Knockoff Filter*

Description

Run the model-free knockoff procedure from start to finish, selecting variables relevant for predicting the outcome of interest.

Usage

```
MFKnockoffs.filter(X, y, knockoffs = MFKnockoffs.create.approximate_gaussian,
  statistic = MFKnockoffs.stat.glmnet_coef_difference, q = 0.1,
  threshold = c("knockoff+", "knockoff"))
```

Arguments

`X`	matrix or data frame of predictors
`y`	response vector
`knockoffs`	the method used to construct knockoffs for the X variables. It must be a function taking a n-by-p matrix X as input and returning a n-by-p matrix of knockoff variables
`statistic`	the test statistic (by default, a lasso statistic with cross validation). See the Details section for more information.
`q`	target FDR (false discovery rate)
`threshold`	either 'knockoff+' or 'knockoff' (default: 'knockoff+').

Details

This function creates the knockoffs, computes the test statistics, and selects variables. It is the main entry point for the model-free knockoff package.

The parameter `knockoffs` controls how knockoff variables are created. By default, a multivariate normal distribution is fitted to the original variables in X. The estimated mean vector and covariance matrix are used to generate second-order approximate Gaussian knockoffs. In general, `knockoffs` should be a function taking a n-by-p matrix of observed variables X and returning a n-by-p matrix of knockoff variables. Two optional functions for creating knockoffs are provided with this package.

If the rows of X are distributed as a multivariate Gaussian with known parameters, then the function `MFKnockoffs.create.gaussian` can be used to generate valid Gaussian knockoff variables, as shown in the examples below.

If the design matrix X is assumed to be fixed instead of random, one can create knockoff variables using the function `MFKnockoffs.create.fixed`. This corresponds to the original framework of the (non Model-Free) knockoff filter.

For more information about creating knockoffs, type `??MFKnockoffs.create`.

The default test statistic is MFKnockoffs.stat.glmnet_coef_difference. For a complete list of the statistics provided with this package, type `??MFKnockoffs.stat`.

It is also possible to provide custom test statistics. An example can be found in the vignette.

Value

An object of class "MFKnockoffs.result". This object is a list containing at least the following components:

`X`	matrix of original variables
`X_k`	matrix of knockoff variables
`statistic`	computed test statistics
`threshold`	computed selection threshold
`selected`	named vector of selected variables

References

Candes et al., Panning for Gold: Model-free Knockoffs for High-dimensional Controlled Variable Selection, arXiv:1610.02351 (2016). https://statweb.stanford.edu/~candes/MF_Knockoffs/index.html

Barber and Candes, Controlling the false discovery rate via knockoffs. Ann. Statist. 43 (2015), no. 5, 2055–2085. https://projecteuclid.org/euclid.aos/1438606853

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

# Basic usage with default arguments
result = MFKnockoffs.filter(X, y)
print(result$selected)

# Advanced usage with custom arguments
knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
k_stat = function(X, X_k, y) MFKnockoffs.stat.glmnet_coef_difference(X, X_k, y, nfolds=5)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.knocks.solve_asdp`

Optimization for SDP knockoffs

Description

Solves the optimization problem needed to create approximate SDP knockoffs

Usage

```
MFKnockoffs.knocks.solve_asdp(Sigma, nBlocks = 10, cores = 1,
  gaptol = 1e-06, maxit = 1000)
```

Arguments

Sigma	A positive-definite correlation matrix
nBlocks	Number of blocks in the block-diagonal approximation of Sigma (default: 10)
cores	Number of cores used to solve the smaller SDPs (default: 1)
gaptol	Tolerance for duality gap as a fraction of the value of the objective functions (default 1e-6)
maxit	The maximum number of iterations for the solver (default: 1000)

Details

Solves the following two-step semidefinite programming problem:

(step 1)

$$\text{maximize } \text{sum}(s) \quad \text{subject to} : 0 <= s <= 1,\ 2\Sigma_{\text{approx}} - \text{diag}(s) >= 0$$

(step 2)

$$\text{maximize } \gamma \quad \text{subject to} : \text{diag}(\gamma s) <= 2\Sigma$$

If the matrix Sigma supplied by the user is a non-scaled covariance matrix (i.e. its diagonal entries are not all equal to 1), then the appropriate scaling is applied before solving the SDP defined above. The result is then scaled back before being returned, as to match the original scaling of the covariance matrix supplied by the user.

Value

The solution s to the semidefinite programming problem defined above

See Also

Other Optimize knockoffs: MFKnockoffs.knocks.solve_equi, MFKnockoffs.knocks.solve_sdp

---

MFKnockoffs.knocks.solve_equi

Optimization for equi-correlated knockoffs

Description

Solves the optimization problem needed to create equi-correlated knockoffs

Usage

```
MFKnockoffs.knocks.solve_equi(Sigma)
```

Arguments

Sigma	A positive-definite covariance matrix

Details

Computes the closed-form solution to the semidefinite programming problem:

$$\text{maximize } s \quad \text{subject to} : 0 <= s <= 1,\ 2\Sigma - sI >= 0$$

used to generate equi-correlated knockoffs.

The closed form-solution to this problem is $s = 2\lambda_{\min}(\Sigma) \wedge 1$

Value

The solution s to the semidefinite programming problem defined above

See Also

Other Optimize knockoffs: MFKnockoffs.knocks.solve_asdp, MFKnockoffs.knocks.solve_sdp

---

MFKnockoffs.knocks.solve_sdp

Optimization for SDP knockoffs

Description

Solves the optimization problem needed to create SDP knockoffs using an interior point method

Usage

```
MFKnockoffs.knocks.solve_sdp(Sigma, gaptol = 1e-06, maxit = 1000)
```

Arguments

Sigma	A positive-definite correlation matrix
gaptol	Tolerance for duality gap as a fraction of the value of the objective functions (default 1e-6)
maxit	The maximum number of iterations for the solver (default: 1000)

Details

Solves the semidefinite programming problem:

$$\text{maximize } \text{sum}(s) \quad \text{subject to} 0 <= s <= 1,\ 2\Sigma - \text{diag}(s) >= 0$$

If the matrix Sigma supplied by the user is a non-scaled covariance matrix (i.e. its diagonal entries are not all equal to 1), then the appropriate scaling is applied before solving the SDP defined above. The result is then scaled back before being returned, as to match the original scaling of the covariance matrix supplied by the user.

Value

The solution s to the semidefinite programming problem defined above

See Also

Other Optimize knockoffs: MFKnockoffs.knocks.solve_asdp, MFKnockoffs.knocks.solve_equi

---

`MFKnockoffs.stat.forward_selection`
Forward selection statistics for MFKnockoffs

Description

Computes the statistic

$$W_j = \max(Z_j, Z_{j+p}) \cdot \mathrm{sgn}(Z_j - Z_{j+p}),$$

where $Z_1, \ldots, Z_{2p}$ give the reverse order in which the 2p variables (the originals and the knockoffs) enter the forward selection model. See the Details for information about forward selection.

Usage

```
MFKnockoffs.stat.forward_selection(X, X_k, y, omp = FALSE)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). It should be numeric
omp	whether to use orthogonal matching pursuit. Default is FALSE

Details

In *forward selection*, the variables are chosen iteratively to maximize the inner product with the residual from the previous step. The initial residual is always `y`. In standard forward selection (`MFKnockoffs.stat.forward_selection`), the next residual is the remainder after regressing on the selected variable; when *orthogonal matching pursuit* is used (`MFKnockoffs.stat.forward_selection_omp`), the next residual is the remainder after regressing on *all* the previously selected variables.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: MFKnockoffs.stat.glmnet_coef_difference, MFKnockoffs.stat.glmnet_lambda_diff
MFKnockoffs.stat.lasso_coef_difference_bin, MFKnockoffs.stat.lasso_coef_difference,
MFKnockoffs.stat.lasso_lambda_difference_bin, MFKnockoffs.stat.lasso_lambda_difference,
MFKnockoffs.stat.random_forest, MFKnockoffs.stat.sqrt_lasso, MFKnockoffs.stat.stability_selection

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

# Basic usage with default arguments
knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.forward_selection)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.forward_selection
k_stat = function(X, X_k, y) foo(X, X_k, y, omp=TRUE)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.glmnet_coef_difference`
Cross-validated GLM statistics for MFKnockoffs

Description

Fit a generalized linear model via penalized maximum likelihood and cross-validation. Then, compute the difference statistic

$$W_j = |Z_j| - |\tilde{Z}_j|$$

where Z_j and $\tilde{Z}_j$ are the coefficient estimates for the jth variable and its knockoff, respectively. The value of the regularization parameter λ is selected by cross-validation and computed with glmnet.

Usage

```
MFKnockoffs.stat.glmnet_coef_difference(X, X_k, y, family = "gaussian",
  cores = 2, ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For family="multinomial", can be a nc>=2 level factor, or

	a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right censored. The function Surv() in package survival produces such a matrix. For family="mgaussian", y is a matrix of quantitative responses.
`family`	Response type (see above)
`cores`	Number of cores used to compute the knockoff statistics by running cv.glmnet. Unless otherwise specified, the number of cores is set equal to two (if available).
`...`	additional arguments specific to 'cv.glmnet' (see Details)

Details

This function uses the `glmnet` package to fit a generalized linear model via penalized maximum likelihood.

The knockoff statistics W_j are constructed by taking the difference between the coefficient of the j-th variable and its knockoff.

By default, the value of the regularization parameter is chosen by 10-fold cross-validation.

The default response family is 'gaussian', for a linear regression model. Different response families (e.g. 'binomial') can be specified by passing an optional parameter 'family'.

The optional `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `100`, where `p` is the number of columns of `X`.

If the family is 'binomial' and a lambda sequence is not provided by the user, this function generates it on a log-linear scale before calling 'glmnet'.

For a complete list of the available additional arguments, see cv.glmnet and glmnet.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: `MFKnockoffs.stat.forward_selection`, `MFKnockoffs.stat.glmnet_lambda_difference`, `MFKnockoffs.stat.lasso_coef_difference_bin`, `MFKnockoffs.stat.lasso_coef_difference`, `MFKnockoffs.stat.lasso_lambda_difference_bin`, `MFKnockoffs.stat.lasso_lambda_difference`, `MFKnockoffs.stat.random_forest`, `MFKnockoffs.stat.sqrt_lasso`, `MFKnockoffs.stat.stability_selection`

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
```

```
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.glmnet_coef_difference)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.glmnet_coef_difference
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.glmnet_lambda_difference`
GLM statistics for MFKnockoffs

Description

Fit a generalized linear model via penalized maximum likelihood and computes the difference statistic

$$W_j = Z_j - \tilde{Z}_j$$

where Z_j and $\tilde{Z}_j$ are the maximum values of the regularization parameter λ at which the jth variable and its knockoff enter the model, respectively.

Usage

```
MFKnockoffs.stat.glmnet_lambda_difference(X, X_k, y, family = "gaussian", ...)
```

Arguments

`X`	original design matrix (size n-by-p)
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n). Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For family="multinomial", can be a nc>=2 level factor, or a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right censored. The function Surv() in package survival produces such a matrix. For family="mgaussian", y is a matrix of quantitative responses.
`family`	Response type (see above)
`...`	additional arguments specific to 'glmnet' (see Details)

Details

This function uses `glmnet` to compute the regularization path on a fine grid of λ's.

The `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `100`.

If the family is 'binomial' and a lambda sequence is not provided by the user, this function generates it on a log-linear scale before calling 'glmnet'.

The default response family is 'gaussian', for a linear regression model. Different response families (e.g. 'binomial') can be specified by passing an optional parameter 'family'.

For a complete list of the available additional arguments, see glmnet.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: `MFKnockoffs.stat.forward_selection`, `MFKnockoffs.stat.glmnet_coef_difference`, `MFKnockoffs.stat.lasso_coef_difference_bin`, `MFKnockoffs.stat.lasso_coef_difference`, `MFKnockoffs.stat.lasso_lambda_difference_bin`, `MFKnockoffs.stat.lasso_lambda_difference`, `MFKnockoffs.stat.random_forest`, `MFKnockoffs.stat.sqrt_lasso`, `MFKnockoffs.stat.stability_selection`

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.glmnet_lambda_difference)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.glmnet_lambda_difference
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.glmnet_lambda_signed_max` *GLM statistics for MFKnockoffs*

Description

Computes the signed maximum statistic

$$W_j = \max(Z_j, \tilde{Z}_j) \cdot \mathrm{sgn}(Z_j - \tilde{Z}_j),$$

where Z_j and $\tilde{Z}_j$ are the maximum values of λ at which the jth variable and its knockoff, respectively, enter the generalized linear model.

Usage

```
MFKnockoffs.stat.glmnet_lambda_signed_max(X, X_k, y, family = "gaussian", ...)
```

Arguments

`X`	original design matrix (size n-by-p)
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n). Quantitative for family="gaussian", or family="poisson" (non-negative counts). For family="binomial" should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). For family="multinomial", can be a nc>=2 level factor, or a matrix with nc columns of counts or proportions. For either "binomial" or "multinomial", if y is presented as a vector, it will be coerced into a factor. For family="cox", y should be a two-column matrix with columns named 'time' and 'status'. The latter is a binary variable, with '1' indicating death, and '0' indicating right censored. The function Surv() in package survival produces such a matrix. For family="mgaussian", y is a matrix of quantitative responses.
`family`	Response type (see above)
`...`	additional arguments specific to 'glmnet' (see Details)

Details

This function uses `glmnet` to compute the regularization path on a fine grid of λ's.

The additional `nlambda` parameter can be used to control the granularity of the grid of λ values. The default value of `nlambda` is `100`.

If the family is 'binomial' and a lambda sequence is not provided by the user, this function generates it on a log-linear scale before calling 'glmnet'.

For a complete list of the available additional arguments, see glmnet.

Value

A vector of statistics W (length p)

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoff=knockoffs,
                            statistic=MFKnockoffs.stat.glmnet_lambda_signed_max)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.glmnet_lambda_signed_max
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

MFKnockoffs.stat.lasso_coef_difference

Cross-validated penalized linear regression statistics for MFKnockoffs

---

Description

Fit a linear regression model via penalized maximum likelihood and cross-validation. Then, compute the difference statistic

$$W_j = |Z_j| - |\tilde{Z}_j|$$

where Z_j and $\tilde{Z}_j$ are the coefficient estimates for the jth variable and its knockoff, respectively. The value of the regularization parameter λ is selected by cross-validation and computed with glmnet.

Usage

```
MFKnockoffs.stat.lasso_coef_difference(X, X_k, y, cores = 2, ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). It should be numeric

cores	Number of cores used to compute the knockoff statistics by running cv.glmnet. If not specified, the number of cores is set to approximately half of the number of cores detected by the parallel package.
...	additional arguments specific to 'glmnet' (see Details)

Details

This function uses the glmnet package to fit the lasso path.

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_coef_difference.

The knockoff statistics W_j are constructed by taking the difference between the coefficient of the j-th variable and its knockoff.

By default, the value of the regularization parameter is chosen by 10-fold cross-validation.

The optional nlambda parameter can be used to control the granularity of the grid of λ's. The default value of nlambda is 100, where p is the number of columns of X.

Unless a lambda sequence is provided by the user, this function generates it on a log-linear scale before calling 'glmnet' (default 'nlambda': 100).

For a complete list of the available additional arguments, see cv.glmnet and glmnet.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: MFKnockoffs.stat.forward_selection, MFKnockoffs.stat.glmnet_coef_difference, MFKnockoffs.stat.glmnet_lambda_difference, MFKnockoffs.stat.lasso_coef_difference_bin, MFKnockoffs.stat.lasso_lambda_difference_bin, MFKnockoffs.stat.lasso_lambda_difference, MFKnockoffs.stat.random_forest, MFKnockoffs.stat.sqrt_lasso, MFKnockoffs.stat.stability_selection

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_coef_difference)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_coef_difference
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.lasso_coef_difference_bin` *Cross-validated penalized logistic regression statistics for MFKnockoffs*

Description

Fit a logistic regression model via penalized maximum likelihood and cross-validation. Then, compute the difference statistic

$$W_j = |Z_j| - |\tilde{Z}_j|$$

where Z_j and $\tilde{Z}_j$ are the coefficient estimates for the jth variable and its knockoff, respectively. The value of the regularization parameter λ is selected by cross-validation and computed with glmnet.

Usage

```
MFKnockoffs.stat.lasso_coef_difference_bin(X, X_k, y, cores = 2, ...)
```

Arguments

`X`	original design matrix (size n-by-p).
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n). It should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). If y is presented as a vector, it will be coerced into a factor.
`cores`	Number of cores used to compute the knockoff statistics by running cv.glmnet. If not specified, the number of cores is set to approximately half of the number of cores detected by the parallel package.
`...`	additional arguments specific to 'glmnet' (see Details)

Details

This function uses the `glmnet` package to fit the penalized logistic regression path.

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_coef_difference.

The knockoff statistics W_j are constructed by taking the difference between the coefficient of the j-th variable and its knockoff.

By default, the value of the regularization parameter is chosen by 10-fold cross-validation.

The optional `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `100`, where `p` is the number of columns of `X`.

For a complete list of the available additional arguments, see cv.glmnet and glmnet.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: MFKnockoffs.stat.forward_selection, MFKnockoffs.stat.glmnet_coef_difference, MFKnockoffs.stat.glmnet_lambda_difference, MFKnockoffs.stat.lasso_coef_difference, MFKnockoffs.stat.lasso_lambda_difference_bin, MFKnockoffs.stat.lasso_lambda_difference, MFKnockoffs.stat.random_forest, MFKnockoffs.stat.sqrt_lasso, MFKnockoffs.stat.stability_selection

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
pr = 1/(1+exp(-X %*% beta))
y = rbinom(n,1,pr)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_coef_difference_bin)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_coef_difference_bin
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

MFKnockoffs.stat.lasso_lambda_difference

Penalized linear regression statistics for MFKnockoffs

Description

Fit the lasso path and computes the difference statistic

$$W_j = Z_j - \tilde{Z}_j$$

where Z_j and $\tilde{Z}_j$ are the maximum values of the regularization parameter λ at which the jth variable and its knockoff enter the penalized linear regression model, respectively.

Usage

```
MFKnockoffs.stat.lasso_lambda_difference(X, X_k, y, ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). It should be numeric.
...	additional arguments specific to 'glmnet' (see Details)

Details

This function uses `glmnet` to compute the lasso path on a fine grid of λ's.

The `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `100`.

Unless a lambda sequence is provided by the user, this function generates it on a log-linear scale before calling 'glmnet' (default 'nlambda': 100).

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_lambda_difference.

For a complete list of the available additional arguments, see glmnet or lars.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: `MFKnockoffs.stat.forward_selection`, `MFKnockoffs.stat.glmnet_coef_difference`, `MFKnockoffs.stat.glmnet_lambda_difference`, `MFKnockoffs.stat.lasso_coef_difference_bin`, `MFKnockoffs.stat.lasso_coef_difference`, `MFKnockoffs.stat.lasso_lambda_difference_bin`, `MFKnockoffs.stat.random_forest`, `MFKnockoffs.stat.sqrt_lasso`, `MFKnockoffs.stat.stability_selection`

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_lambda_difference)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_lambda_difference
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.lasso_lambda_difference_bin`
Penalized logistic regression statistics for MFKnockoffs

Description

Fit the lasso path and computes the difference statistic

$$W_j = Z_j - \tilde{Z}_j$$

where Z_j and $\tilde{Z}_j$ are the maximum values of the regularization parameter λ at which the jth variable and its knockoff enter the penalized logistic regression model, respectively.

Usage

```
MFKnockoffs.stat.lasso_lambda_difference_bin(X, X_k, y, ...)
```

Arguments

`X`	original design matrix (size n-by-p)
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n). It should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). If y is presented as a vector, it will be coerced into a factor.
`...`	additional arguments specific to 'glmnet' (see Details)

Details

This function uses `glmnet` to compute the lasso path on a fine grid of λ's.

The `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `100`.

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_lambda_difference.

For a complete list of the available additional arguments, see glmnet or lars.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: `MFKnockoffs.stat.forward_selection`, `MFKnockoffs.stat.glmnet_coef_difference`, `MFKnockoffs.stat.glmnet_lambda_difference`, `MFKnockoffs.stat.lasso_coef_difference_bin`, `MFKnockoffs.stat.lasso_coef_difference`, `MFKnockoffs.stat.lasso_lambda_difference`, `MFKnockoffs.stat.random_forest`, `MFKnockoffs.stat.sqrt_lasso`, `MFKnockoffs.stat.stability_selection`

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
pr = 1/(1+exp(-X %*% beta))
y = rbinom(n,1,pr)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_lambda_difference_bin)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_lambda_difference_bin
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

MFKnockoffs.stat.lasso_lambda_signed_max

Penalized linear regression statistics for MFKnockoffs

Description

Computes the signed maximum statistic

$$W_j = \max(Z_j, \tilde{Z}_j) \cdot \mathrm{sgn}(Z_j - \tilde{Z}_j),$$

where Z_j and $\tilde{Z}_j$ are the maximum values of λ at which the jth variable and its knockoff, respectively, enter the penalized linear regression model.

Usage

```
MFKnockoffs.stat.lasso_lambda_signed_max(X, X_k, y, ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). It should be numeric.
...	additional arguments specific to 'glmnet' or 'lars' (see Details)

Details

This function uses `glmnet` to compute the regularization path on a fine grid of λ's.

The additional `nlambda` parameter can be used to control the granularity of the grid of λ values. The default value of `nlambda` is `100`.

Unless a lambda sequence is provided by the user, this function generates it on a log-linear scale before calling 'glmnet' (default 'nlambda': 100).

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_lambda_difference.

For a complete list of the available additional arguments, see glmnet.

Value

A vector of statistics W (length p)

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoff=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_lambda_signed_max)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_lambda_signed_max
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

MFKnockoffs.stat.lasso_lambda_signed_max_bin

Penalized logistic regression statistics for MFKnockoffs

Description

Computes the signed maximum statistic

$$W_j = \max(Z_j, \tilde{Z}_j) \cdot \mathrm{sgn}(Z_j - \tilde{Z}_j),$$

where Z_j and $\tilde{Z}_j$ are the maximum values of λ at which the jth variable and its knockoff, respectively, enter the penalized logistic regression model.

Usage

```
MFKnockoffs.stat.lasso_lambda_signed_max_bin(X, X_k, y, ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n). It should be either a factor with two levels, or a two-column matrix of counts or proportions (the second column is treated as the target class; for a factor, the last level in alphabetical order is the target class). If y is presented as a vector, it will be coerced into a factor.
...	additional arguments specific to 'glmnet' or 'lars' (see Details)

Details

This function uses `glmnet` to compute the regularization path on a fine grid of λ's.

The additional `nlambda` parameter can be used to control the granularity of the grid of λ values. The default value of `nlambda` is `100`.

This function is a wrapper around the more general MFKnockoffs.stat.glmnet_lambda_difference.

For a complete list of the available additional arguments, see glmnet.

Value

A vector of statistics W (length p)

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
pr = 1/(1+exp(-X %*% beta))
y = rbinom(n,1,pr)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoff=knockoffs,
                            statistic=MFKnockoffs.stat.lasso_lambda_signed_max_bin)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.lasso_lambda_signed_max_bin
k_stat = function(X, X_k, y) foo(X, X_k, y, nlambda=200)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.random_forest` *Random forest statistics for MFKnockoffs*

Description

Computes the difference statistic

$$W_j = |Z_j| - |\tilde{Z}_j|$$

where Z_j and $\tilde{Z}_j$ are the random forest feature importances of the jth variable and its knockoff, respectively.

Usage

```
MFKnockoffs.stat.random_forest(X, X_k, y, ...)
```

Arguments

`X`	original design matrix (size n-by-p)
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n). If a factor, classification is assumed, otherwise regression is assumed.
`...`	additional arguments specific to 'ranger' (see Details)

Details

This function uses the `ranger` package to compute variable importance measures. The importance of a variable is measured as the total decrease in node impurities from splitting on that variable, averaged over all trees. For regression, the node impurity is measured by residual sum of squares. For classification, it is measured by the Gini index.

For a complete list of the available additional arguments, see ranger.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: `MFKnockoffs.stat.forward_selection`, `MFKnockoffs.stat.glmnet_coef_difference`, `MFKnockoffs.stat.glmnet_lambda_difference`, `MFKnockoffs.stat.lasso_coef_difference_bin`, `MFKnockoffs.stat.lasso_coef_difference`, `MFKnockoffs.stat.lasso_lambda_difference_bin`, `MFKnockoffs.stat.lasso_lambda_difference`, `MFKnockoffs.stat.sqrt_lasso`, `MFKnockoffs.stat.stability_sel`

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.random_forest)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.random_forest
k_stat = function(X, X_k, y) foo(X, X_k, y, nodesize=5)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.stat.sqrt_lasso`

SQRT-lasso statistics for Knockoff

Description

Computes the signed maximum statistic

$$W_j = \max(Z_j, \tilde{Z}_j) \cdot \mathrm{sgn}(Z_j - \tilde{Z}_j),$$

where Z_j and $\tilde{Z}_j$ are the maximum values of λ at which the jth variable and its knockoff, respectively, enter the SQRT lasso model.

Usage

```
MFKnockoffs.stat.sqrt_lasso(X, X_k, y, ...)
```

Arguments

`X`	original design matrix (size n-by-p)
`X_k`	knockoff matrix (size n-by-p)
`y`	response vector (length n) of numeric type
`...`	additional arguments specific to 'slim'

Details

With default parameters, this function uses the package `flare` to run the SQRT lasso. By specifying the appropriate optional parameters, one can use different Lasso variants including Dantzig Selector, LAD Lasso, SQRT Lasso and Lq Lasso for estimating high dimensional sparse linear models.

For a complete list of the available additional arguments, see slim.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: MFKnockoffs.stat.forward_selection, MFKnockoffs.stat.glmnet_coef_difference, MFKnockoffs.stat.glmnet_lambda_difference, MFKnockoffs.stat.lasso_coef_difference_bin, MFKnockoffs.stat.lasso_coef_difference, MFKnockoffs.stat.lasso_lambda_difference_bin, MFKnockoffs.stat.lasso_lambda_difference, MFKnockoffs.stat.random_forest, MFKnockoffs.stat.stability_s

Examples

```
p=50; n=50; k=10
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=MFKnockoffs.stat.sqrt_lasso)
print(result$selected)

# Advanced usage with custom arguments
foo = MFKnockoffs.stat.sqrt_lasso
k_stat = function(X, X_k, y) foo(X, X_k, y, q=0.5)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

MFKnockoffs.stat.stability_selection

Stability selection statistics for MFKnockoffs

Description

Computes the difference statistic

$$W_j = |Z_j| - |\tilde{Z}_j|$$

where Z_j and $\tilde{Z}_j$ are measure the importance of the jth variable and its knockoff, respectively, based on the stability of their selection upon subsampling of the data.

Usage

```
MFKnockoffs.stat.stability_selection(X, X_k, y, fitfun = stabs::glmnet.lasso,
  ...)
```

Arguments

X	original design matrix (size n-by-p)
X_k	knockoff matrix (size n-by-p)
y	response vector (length n)
fitfun	fitfun a function that takes the arguments x, y as above, and additionally the number of variables to include in each model q. The function then needs to fit the model and to return a logical vector that indicates which variable was selected (among the q selected variables). The name of the function should be prefixed by 'stabs::'.
...	additional arguments specific to 'stabs' (see Details)

Details

This function uses the `stabs` package to compute variable selection stability. The selection stability of the j-th variable is defined as its probability of being selected upon random subsampling of the data. The default method for selecting variables in each subsampled dataset is `stabs::glmnet.lasso_maxCoef`.

For a complete list of the available additional arguments, see stabsel.

Value

A vector of statistics W (length p)

See Also

Other statistics for knockoffs: MFKnockoffs.stat.forward_selection, MFKnockoffs.stat.glmnet_coef_difference, MFKnockoffs.stat.glmnet_lambda_difference, MFKnockoffs.stat.lasso_coef_difference_bin, MFKnockoffs.stat.lasso_coef_difference, MFKnockoffs.stat.lasso_lambda_difference_bin, MFKnockoffs.stat.lasso_lambda_difference, MFKnockoffs.stat.random_forest, MFKnockoffs.stat.sqrt_lasso

Examples

```
p=100; n=200; k=15
mu = rep(0,p); Sigma = diag(p)
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

knockoffs = function(X) MFKnockoffs.create.gaussian(X, mu, Sigma)
# Basic usage with default arguments
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs,
                            statistic=MFKnockoffs.stat.stability_selection)
print(result$selected)
```

```
# Advanced usage with custom arguments
foo = MFKnockoffs.stat.stability_selection
k_stat = function(X, X_k, y) foo(X, X_k, y, fitfun=stabs::lars.lasso)
result = MFKnockoffs.filter(X, y, knockoffs=knockoffs, statistic=k_stat)
print(result$selected)
```

`MFKnockoffs.threshold` *Threshold for the knockoff filter*

Description

Computes the threshold for the knockoff filter.

Usage

```
MFKnockoffs.threshold(W, q = 0.1, method = c("knockoff+", "knockoff"))
```

Arguments

`W`	the test statistics
`q`	target FDR (false discovery rate)
`method`	either 'knockoff or 'knockoff+'

Value

The threshold for variable selection.

`print.MFKnockoffs.result`
Print results for the knockoff filter

Description

Prints the list of variables selected by the knockoff filter and the corresponding function call.

Usage

```
## S3 method for class 'MFKnockoffs.result'
print(x, ...)
```

Arguments

`x`	the output of a call to MFKnockoffs.filter
`...`	unused

Index