Package ‘knockoff’

June 6, 2020

Type Package

Title The Knockoff Filter for Controlled Variable Selection

Version 0.3.3

Date 2020-06-01

Description The knockoff filter is 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-X knockoffs for high-dimensional controlled variable selection'', J. R. Statist. Soc. B (2018) 80, 3, pp. 551-577.

License GPL-3

URL https://web.stanford.edu/group/candes/knockoffs/index.html

Depends methods, stats

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

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

RoxygenNote 7.1.0

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Matteo Sesia <sesia@marshall.usc.edu>

Repository CRAN

Date/Publication 2020-06-06 05:00:02 UTC

R topics documented:

`create.fixed` *Fixed-X knockoffs*

Description

Creates fixed-X knockoff variables.

Usage

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

Arguments

`X`	normalized n-by-p matrix of original variables.($n \geq p$).
`method`	either "equi" or "sdp" (default: "sdp"). This determines the method that will be used to minimize the correlation between the original variables and the knockoffs.
`sigma`	the noise level, used to augment the data with extra rows if necessary (default: NULL).
`y`	vector of length n, containing the observed responses. This is needed to estimate the noise level if the parameter `sigma` is not provided, in case $p \leq n < 2p$ (default: NULL).
`randomize`	whether the knockoffs are constructed deterministically or randomized (default: F).

Details

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

Value

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

`X`	n-by-p matrix of original variables (possibly augmented or transformed).
`Xk`	n-by-p matrix of knockoff variables.
`y`	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

See Also

Other create: `create.gaussian()`, `create.second_order()`

Examples

```
p=100; n=200; k=15
X = matrix(rnorm(n*p),n)
nonzero = sample(p, k)
beta = 5.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

# Basic usage with default arguments
result = knockoff.filter(X, y, knockoffs=create.fixed)
print(result$selected)
```

```
# Advanced usage with custom arguments
knockoffs = function(X) create.fixed(X, method='equi')
result = knockoff.filter(X, y, knockoffs=knockoffs)
print(result$selected)
```

`create.gaussian` *Model-X Gaussian knockoffs*

Description

Samples multivariate Gaussian model-X knockoff variables.

Usage

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

Arguments

`X`	n-by-p matrix of original variables.
`mu`	vector of length p, indicating the mean parameter of the Gaussian model for X.
`Sigma`	p-by-p covariance matrix for the Gaussian model of X.
`method`	either "equi", "sdp" or "asdp" (default: "asdp"). This determines the method that will be used to minimize the correlation between the original variables and the knockoffs.
`diag_s`	vector of length p, containing the pre-computed covariances between the original variables and the knockoffs. This will be computed according to `method`, if not supplied.

Value

A 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://web.stanford.edu/group/candes/knockoffs/index.html

See Also

Other create: `create.fixed()`, `create.second_order()`

Examples

```
p=200; n=100; k=15
rho = 0.4
mu = rep(0,p); Sigma = toeplitz(rho^(0:(p-1)))
X = matrix(rnorm(n*p),n) %*% chol(Sigma)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

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

# Advanced usage with custom arguments
knockoffs = function(X) create.gaussian(X, mu, Sigma, method='equi')
result = knockoff.filter(X, y, knockoffs=knockoffs)
print(result$selected)
```

---

create.second_order   *Second-order Gaussian knockoffs*

Description

This function samples second-order multivariate Gaussian knockoff variables. First, a multivariate Gaussian distribution is fitted to the observations of X. Then, Gaussian knockoffs are generated according to the estimated model.

Usage

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

Arguments

X	n-by-p matrix of original variables.
method	either "equi", "sdp" or "asdp" (default: "asdp"). This determines the method that will be used to minimize the correlation between the original variables and the knockoffs.
shrink	whether to shrink the estimated covariance matrix (default: F).

Details

If the argument `shrink` is set to T, 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`. See `cov.shrink` for more details.

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

Value

A 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://web.stanford.edu/group/candes/knockoffs/index.html

See Also

Other create: `create.fixed()`, `create.gaussian()`

Examples

```
p=200; n=100; k=15
rho = 0.4
Sigma = toeplitz(rho^(0:(p-1)))
X = matrix(rnorm(n*p),n) %*% chol(Sigma)
nonzero = sample(p, k)
beta = 3.5 * (1:p %in% nonzero)
y = X %*% beta + rnorm(n)

# Basic usage with default arguments
result = knockoff.filter(X, y, knockoffs=create.second_order)
print(result$selected)

# Advanced usage with custom arguments
knockoffs = function(X) create.second_order(X, method='equi')
result = knockoff.filter(X, y, knockoffs=knockoffs)
print(result$selected)
```

`create.solve_asdp` *Relaxed optimization for fixed-X and Gaussian knockoffs*

Description

This function solves the optimization problem needed to create fixed-X and Gaussian SDP knockoffs on a block-diagonal approximation of the covariance matrix. This will be less powerful than `create.solve_sdp`, but more computationally efficient.

Usage

```
create.solve_asdp(
  Sigma,
  max.size = 500,
  gaptol = 1e-06,
  maxit = 1000,
  verbose = FALSE
)
```

Arguments

`Sigma`	positive-definite p-by-p covariance matrix.
`max.size`	size of the largest block in the block-diagonal approximation of Sigma (default: 500). See Details.
`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).
`verbose`	whether to display progress (default: FALSE).

Details

Solves the following two-step semidefinite program:

(step 1)

$$\text{maximize } \mathrm{sum}(s) \quad \text{subject to} : 0 \leq s \leq 1, \; 2\Sigma_{\text{approx}} - \mathrm{diag}(s) \geq 0$$

(step 2)

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

Each smaller SDP is solved using the interior-point method implemented in `dsdp`.

The parameter max.size controls the size of the largest semidefinite program that needs to be solved. A larger value of max.size will increase the computation cost, while yielding a solution closer to that of the original semidefinite program.

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 program defined above.

See Also

Other optimization: `create.solve_equi()`, `create.solve_sdp()`

`create.solve_equi` *Optimization for equi-correlated fixed-X and Gaussian knockoffs*

Description

This function solves a very simple optimization problem needed to create fixed-X and Gaussian SDP knockoffs on the full the covariance matrix. This may be significantly less powerful than `create.solve_sdp`.

Usage

```
create.solve_equi(Sigma)
```

Arguments

`Sigma`	positive-definite p-by-p covariance matrix.

Details

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

$$\text{maximize } s \quad \text{subject to} : 0 \leq s \leq 1, \; 2\Sigma - sI \geq 0$$

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 optimization problem defined above.

See Also

Other optimization: `create.solve_asdp()`, `create.solve_sdp()`

`create.solve_sdp` *Optimization for fixed-X and Gaussian knockoffs*

Description

This function solves the optimization problem needed to create fixed-X and Gaussian SDP knockoffs on the full covariance matrix. This will be more powerful than `create.solve_asdp`, but more computationally expensive.

Usage

```
create.solve_sdp(Sigma, gaptol = 1e-06, maxit = 1000, verbose = FALSE)
```

Arguments

`Sigma`	positive-definite p-by-p covariance matrix.
`gaptol`	tolerance for duality gap as a fraction of the value of the objective functions (default: 1e-6).
`maxit`	maximum number of iterations for the solver (default: 1000).
`verbose`	whether to display progress (default: FALSE).

Details

Solves the semidefinite programming problem:

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

This problem is solved using the interior-point method implemented in `dsdp`.

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 optimization: `create.solve_asdp()`, `create.solve_equi()`

`knockoff` *knockoff: A package for controlled variable selection*

Description

This package implements the Knockoff Filter, which is a powerful and versatile tool for controlled variable selection.

Outline

The procedure is based on the contruction of artificial 'knockoff copies' of the variables present in the given statistical model. Then, it selects those variables that are clearly better than their corresponding knockoffs, based on some measure of variable importance. A wide range of statistics and machine learning tools can be exploited to estimate the importance of each variable, while guaranteeing finite-sample control of the false discovery rate (FDR).

The Knockoff Filter controls the FDR in either of two statistical scenarios:

- The "model-X" scenario: the response Y can depend on the variables $X = (X_1, \ldots, X_p)$ in an arbitrary and unknown fashion, but the distribution of X must be known. In thise case there are no constraints on the dimensions n and p of the problem.
- The "fixed-X" scenario: the response Y depends upon X through a homoscedastic Gaussian linear model and the problem is low-dimensional ($n \geq p$). In this case, no modeling assumptions on X are required.

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

https://web.stanford.edu/group/candes/knockoffs/index.html knockoff.filter *The Knockoff Filter*

Description

This function runs the Knockoffs procedure from start to finish, selecting variables relevant for predicting the outcome of interest.

Usage

```
knockoff.filter(
  X,
  y,
  knockoffs = create.second_order,
  statistic = stat.glmnet_coefdiff,
  fdr = 0.1,
  offset = 1
)
```

Arguments

X	n-by-p matrix or data frame of predictors.
y	response vector of length n.
knockoffs	method used to construct knockoffs for the X variables. It must be a function taking a n-by-p matrix as input and returning a n-by-p matrix of knockoff variables. By default, approximate model-X Gaussian knockoffs are used.
statistic	statistics used to assess variable importance. By default, a lasso statistic with cross-validation is used. See the Details section for more information.
fdr	target false discovery rate (default: 0.1).
offset	either 0 or 1 (default: 1). This is the offset used to compute the rejection threshold on the statistics. The value 1 yields a slightly more conservative procedure ("knockoffs+") that controls the false discovery rate (FDR) according to the usual definition, while an offset of 0 controls a modified FDR.

Details

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

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

In the model-X scenario, under the assumption that the rows of X are distributed as a multivariate Gaussian with known parameters, then the function `create.gaussian` can be used to generate Gaussian knockoffs, as shown in the examples below.

In the fixed-X scenario, one can create the knockoffs using the function `create.fixed`. This requires $n \geq p$ and it assumes that the response Y follows a homoscedastic linear regression model.

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

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

It is possible to provide custom functions for the knockoff constructions or the importance statistics. Some examples can be found in the vignette.

Value

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

`X`	matrix of original variables
`Xk`	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://web.stanford.edu/group/candes/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=200; n=100; 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 = knockoff.filter(X, y)
print(result$selected)

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

knockoff.threshold *Threshold for the knockoff filter*

Description

Computes the threshold for the knockoff filter.

Usage

```
knockoff.threshold(W, fdr = 0.1, offset = 1)
```

Arguments

W	the test statistics
fdr	target false discovery rate (default: 0.1)
offset	either 0 or 1 (default: 1). The offset used to compute the rejection threshold on the statistics. The value 1 yields a slightly more conservative procedure ("knockoffs+") that controls the FDR according to the usual definition, while an offset of 0 controls a modified FDR.

Value

The threshold for variable selection.

print.knockoff.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 'knockoff.result'
print(x, ...)
```

Arguments

x	the output of a call to knockoff.filter
...	unused

`stat.forward_selection` *Importance statistics based on forward selection*

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

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

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	numeric vector of length n, containing the response variables.
`omp`	whether to use orthogonal matching pursuit (default: F).

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 (`stat.forward_selection`), the next residual is the remainder after regressing on the selected variable; when orthogonal matching pursuit is used, the next residual is the remainder after regressing on *all* the previously selected variables.

Value

A vector of statistics W of length p.

See Also

Other statistics: `stat.glmnet_coefdiff()`, `stat.glmnet_lambdadiff()`, `stat.lasso_coefdiff_bin()`, `stat.lasso_coefdiff()`, `stat.lasso_lambdadiff_bin()`, `stat.lasso_lambdadiff()`, `stat.random_forest()`, `stat.sqrt_lasso()`, `stat.stability_selection()`

Examples

```
p=100; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

---

stat.glmnet_coefdiff *Importance statistics based on a GLM with cross-validation*

Description

Fits 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

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

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. 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 statistics by running cv.glmnet. Unless otherwise specified, the number of cores is set equal to two (if available).

`...` additional arguments specific to `glmnet` (see Details).

Details

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

The 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 `500`, 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 of length p.

See Also

Other statistics: `stat.forward_selection()`, `stat.glmnet_lambdadiff()`, `stat.lasso_coefdiff_bin()`, `stat.lasso_coefdiff()`, `stat.lasso_lambdadiff_bin()`, `stat.lasso_lambdadiff()`, `stat.random_forest()`, `stat.sqrt_lasso()`, `stat.stability_selection()`

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

# Basic usage with default arguments
result = knockoff.filter(X, y, knockoffs=knockoffs,
```

```
                              statistic=stat.glmnet_coefdiff)
print(result$selected)

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

---

stat.glmnet_lambdadiff

Importance statistics based on a GLM

Description

Fits 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

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

Arguments

X	n-by-p matrix of original variables.
X_k	n-by-p matrix of knockoff variables.
y	vector of length n, containing the response variables. 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 500.

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 of length p.

See Also

Other statistics: stat.forward_selection(), stat.glmnet_coefdiff(), stat.lasso_coefdiff_bin(), stat.lasso_coefdiff(), stat.lasso_lambdadiff_bin(), stat.lasso_lambdadiff(), stat.random_forest(), stat.sqrt_lasso(), stat.stability_selection()

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

`stat.glmnet_lambdasmax` *GLM 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 generalized linear model.

Usage

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

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. 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 `500`.

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 of length p.

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

`stat.lasso_coefdiff` *Importance statistics based the lasso with cross-validation*

Description

Fits 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

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

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. It should be numeric
`cores`	Number of cores used to compute the 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 and is a wrapper around the more general stat.glmnet_coefdiff.

The 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 500, 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': 500).

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

Value

A vector of statistics W of length p.

See Also

Other statistics: stat.forward_selection(), stat.glmnet_coefdiff(), stat.glmnet_lambdadiff(), stat.lasso_coefdiff_bin(), stat.lasso_lambdadiff_bin(), stat.lasso_lambdadiff(), stat.random_forest(), stat.sqrt_lasso(), stat.stability_selection()

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

stat.lasso_coefdiff_bin   *Importance statistics based on regularized logistic regression with cross-validation*

Description

Fits 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

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

Arguments

`X`	n-by-p matrix of original variables..
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. 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 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 and is a wrapper around the more general `stat.glmnet_coefdiff`.

The 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 `500`, 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 of length p.

See Also

Other statistics: stat.forward_selection(), stat.glmnet_coefdiff(), stat.glmnet_lambdadiff(), stat.lasso_coefdiff(), stat.lasso_lambdadiff_bin(), stat.lasso_lambdadiff(), stat.random_forest(), stat.sqrt_lasso(), stat.stability_selection()

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

---

stat.lasso_lambdadiff *Importance statistics based on the lasso*

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

```
stat.lasso_lambdadiff(X, X_k, y, ...)
```

Arguments

X	n-by-p matrix of original variables.
X_k	n-by-p matrix of knockoff variables.
y	vector of length n, containing the response variables. 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 and is a wrapper around the more general stat.glmnet_lambdadiff.

The `nlambda` parameter can be used to control the granularity of the grid of λ's. The default value of `nlambda` is `500`.

Unless a lambda sequence is provided by the user, this function generates it on a log-linear scale before calling `glmnet` (default 'nlambda': 500).

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

Value

A vector of statistics W of length p.

See Also

Other statistics: `stat.forward_selection()`, `stat.glmnet_coefdiff()`, `stat.glmnet_lambdadiff()`, `stat.lasso_coefdiff_bin()`, `stat.lasso_coefdiff()`, `stat.lasso_lambdadiff_bin()`, `stat.random_forest()`, `stat.sqrt_lasso()`, `stat.stability_selection()`

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

`stat.lasso_lambdadiff_bin` *Importance statistics based on regularized logistic regression*

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

```
stat.lasso_lambdadiff_bin(X, X_k, y, ...)
```

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. 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 `500`.

This function is a wrapper around the more general `stat.glmnet_lambdadiff`.

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

Value

A vector of statistics W of length p.

See Also

Other statistics: `stat.forward_selection()`, `stat.glmnet_coefdiff()`, `stat.glmnet_lambdadiff()`, `stat.lasso_coefdiff_bin()`, `stat.lasso_coefdiff()`, `stat.lasso_lambdadiff()`, `stat.random_forest()`, `stat.sqrt_lasso()`, `stat.stability_selection()`

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

---

stat.lasso_lambdasmax *Penalized linear regression 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 penalized linear regression model.

Usage

```
stat.lasso_lambdasmax(X, X_k, y, ...)
```

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables. 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 `500`.

Unless a lambda sequence is provided by the user, this function generates it on a log-linear scale before calling `glmnet` (default 'nlambda': 500).

This function is a wrapper around the more general `stat.glmnet_lambdadiff`.

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

Value

A vector of statistics W of length p.

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

---

stat.lasso_lambdasmax_bin

Penalized logistic regression 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 penalized logistic regression model.

Usage

```
stat.lasso_lambdasmax_bin(X, X_k, y, ...)
```

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.

y	vector of length n, containing the response variables. 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 `500`.

This function is a wrapper around the more general stat.glmnet_lambdadiff.

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

Value

A vector of statistics W of length p.

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

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

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

`stat.random_forest` *Importance statistics based on random forests*

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

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

Arguments

X	n-by-p matrix of original variables.
X_k	n-by-p matrix of knockoff variables.
y	vector of length n, containing the response variables. 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 of length p.

See Also

Other statistics: stat.forward_selection(), stat.glmnet_coefdiff(), stat.glmnet_lambdadiff(), stat.lasso_coefdiff_bin(), stat.lasso_coefdiff(), stat.lasso_lambdadiff_bin(), stat.lasso_lambdadiff(), stat.sqrt_lasso(), stat.stability_selection()

Examples

```
p=200; n=100; 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) create.gaussian(X, mu, Sigma)

# Basic usage with default arguments
result = knockoff.filter(X, y, knockoffs=knockoffs,
```

```
                            statistic=stat.random_forest)
print(result$selected)

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

`stat.sqrt_lasso` *Importance statistics based on the square-root lasso*

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

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

Arguments

`X`	n-by-p matrix of original variables.
`X_k`	n-by-p matrix of knockoff variables.
`y`	vector of length n, containing the response variables of numeric type.
`...`	additional arguments specific to `slim`.

Details

With default parameters, this function uses the package `RPtests` 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 `sqrt_lasso`.

Value

A vector of statistics W of length p.

See Also

Other statistics: stat.forward_selection(), stat.glmnet_coefdiff(), stat.glmnet_lambdadiff(), stat.lasso_coefdiff_bin(), stat.lasso_coefdiff(), stat.lasso_lambdadiff_bin(), stat.lasso_lambdadiff(), stat.random_forest(), stat.stability_selection()

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) create.gaussian(X, mu, Sigma)

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

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

---

stat.stability_selection

Importance statistics based on stability selection

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

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

Arguments

X	n-by-p matrix of original variables.
X_k	n-by-p matrix of knockoff variables.
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 `lars.lasso`.

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

Value

A vector of statistics W of length p.

See Also

Other statistics: `stat.forward_selection()`, `stat.glmnet_coefdiff()`, `stat.glmnet_lambdadiff()`, `stat.lasso_coefdiff_bin()`, `stat.lasso_coefdiff()`, `stat.lasso_lambdadiff_bin()`, `stat.lasso_lambdadiff()`, `stat.random_forest()`, `stat.sqrt_lasso()`

Examples

```
p=50; n=50; 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) create.gaussian(X, mu, Sigma)

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

Index