

# Package: SDModels (via r-universe)

March 11, 2025

**Title** Spectrally Deconfounded Models

**Version** 1.0.6

**Description** Screen for and analyze non-linear sparse direct effects in the presence of unobserved confounding using the spectral deconfounding techniques (Ćevic, Bühlmann, and Meinshausen (2020) <jmlr.org/papers/v21/19-545.html>, Guo, Ćevic, and Bühlmann (2022) <doi:10.1214/21-AOS2152>). These methods have been shown to be a good estimate for the true direct effect if we observe many covariates, e.g., high-dimensional settings, and we have fairly dense confounding. Even if the assumptions are violated, it seems like there is not much to lose, and the deconfounded models will, in general, estimate a function closer to the true one than classical least squares optimization. 'SDModels' provides functions SDAM() for Spectrally Deconfounded Additive Models (Scheidegger, Guo, and Bühlmann (2025) <doi:10.1145/3711116>) and SDForest() for Spectrally Deconfounded Random Forests (Ulmer, Scheidegger, and Bühlmann (2025) <doi:10.48550/arXiv.2502.03969>).

**License** GPL-3

**Imports** data.tree, DiagrammeR, doParallel, ggplot2, GPUmatrix, gridExtra, locatexec, parallel, pbapply, Rdpack, tidyr, fda, grplasso, rlang

**Suggests** plotly, datasets, rpart, knitr, rmarkdown, ranger, HDclassif, qpdf, igraph, testthat (>= 3.0.0)

**RdMacros** Rdpack

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**VignetteBuilder** knitr

**URL** <https://markusul.github.io/SDModels/>

**BugReports** <https://github.com/markusul/SDModels/issues>

**Config/testthat/edition** 3

**Config/pak/sysreqs** libgplk-dev make libicu-dev libxml2-dev libx11-dev

**Repository** <https://markusul.r-universe.dev>  
**RemoteUrl** <https://github.com/markusul/sdmodels>  
**RemoteRef** HEAD  
**RemoteSha** ba5caaafd43ff57074fff84b96c565693f025135

## Contents

|                                       |    |
|---------------------------------------|----|
| copy.SDForest . . . . .               | 3  |
| copy.SDTree . . . . .                 | 4  |
| cvSDTree . . . . .                    | 5  |
| fromList.SDForest . . . . .           | 7  |
| fromList.SDTree . . . . .             | 8  |
| f_four . . . . .                      | 9  |
| get_cp_seq.SDForest . . . . .         | 10 |
| get_cp_seq.SDTree . . . . .           | 11 |
| get_Q . . . . .                       | 12 |
| get_W . . . . .                       | 13 |
| mergeForest . . . . .                 | 14 |
| partDependence . . . . .              | 15 |
| plot.partDependence . . . . .         | 16 |
| plot.paths . . . . .                  | 17 |
| plot.SDTree . . . . .                 | 18 |
| plotOOB . . . . .                     | 18 |
| predict.SDAM . . . . .                | 19 |
| predict.SDForest . . . . .            | 20 |
| predict.SDTree . . . . .              | 21 |
| predictOOB . . . . .                  | 22 |
| predict_individual_fj . . . . .       | 23 |
| print.partDependence . . . . .        | 24 |
| print.SDAM . . . . .                  | 25 |
| print.SDForest . . . . .              | 26 |
| print.SDTree . . . . .                | 27 |
| prune.SDForest . . . . .              | 28 |
| prune.SDTree . . . . .                | 29 |
| regPath.SDForest . . . . .            | 30 |
| regPath.SDTree . . . . .              | 31 |
| SDAM . . . . .                        | 32 |
| SDForest . . . . .                    | 35 |
| SDTree . . . . .                      | 40 |
| simulate_data_nonlinear . . . . .     | 43 |
| simulate_data_step . . . . .          | 45 |
| stabilitySelection.SDForest . . . . . | 46 |
| toList.SDForest . . . . .             | 47 |
| toList.SDTree . . . . .               | 48 |
| varImp.SDAM . . . . .                 | 49 |
| varImp.SDForest . . . . .             | 50 |
| varImp.SDTree . . . . .               | 51 |

---

|               |                      |
|---------------|----------------------|
| copy.SDForest | <i>Copy a forest</i> |
|---------------|----------------------|

---

**Description**

Returns a copy of the forest object. Might be useful if you want to keep the original forest in comparison to the pruned forest.

**Usage**

```
## S3 method for class 'SDForest'  
copy(object, ...)
```

**Arguments**

|        |  |
|--------|--|
| object | an SDForest object                                 |
| ...    | Further arguments passed to or from other methods. |

**Value**

A copy of the SDForest object

**Author(s)**

Markus Ulmer

**See Also**

[prune](#)

**Examples**

```
set.seed(1)  
X <- matrix(rnorm(10 * 20), nrow = 10)  
Y <- rnorm(10)  
fit <- SDForest(x = X, y = Y, nTree = 2, cp = 0.5)  
fit2 <- copy(fit)
```

---

`copy.SDTree`*Copy a tree*

---

**Description**

Returns a copy of the tree object. Might be useful if you want to keep the original tree in comparison to the pruned tree.

**Usage**

```
## S3 method for class 'SDTree'  
copy(object, ...)
```

**Arguments**

|                     |  |
|---------------------|--|
| <code>object</code> | an SDTree object                                   |
| <code>...</code>    | Further arguments passed to or from other methods. |

**Value**

A copy of the SDTree object

**Author(s)**

Markus Ulmer

**See Also**

[prune](#)

**Examples**

```
set.seed(1)  
X <- matrix(rnorm(10 * 20), nrow = 10)  
Y <- rnorm(10)  
fit <- SDTree(x = X, y = Y, cp = 0.5)  
fit2 <- copy(fit)
```

---

cvSDTree

*Cross-validation for the SDTree*


---

## Description

Estimates the optimal complexity parameter for the SDTree using cross-validation. The transformations are estimated for each training set and validation set separately to ensure independence of the validation set.

## Usage

```
cvSDTree(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  max_leaves = NULL,
  cp = 0,
  min_sample = 5,
  mtry = NULL,
  fast = TRUE,
  Q_type = "trim",
  trim_quantile = 0.5,
  q_hat = 0,
  Qf = NULL,
  A = NULL,
  gamma = 0.5,
  gpu = FALSE,
  mem_size = 1e+07,
  max_candidates = 100,
  nfolds = 3,
  cp_seq = NULL,
  mc.cores = 1,
  Q_scale = TRUE
)
```

## Arguments

|            |  |
|------------|--|
| formula    | Object of class formula or describing the model to fit of the form $y \sim x_1 + x_2 + \dots$ where $y$ is a numeric response and $x_1, x_2, \dots$ are vectors of covariates. Interactions are not supported. |
| data       | Training data of class data.frame containing the variables in the model.   |
| x          | Predictor data, alternative to formula and data.   |
| y          | Response vector, alternative to formula and data.  |
| max_leaves | Maximum number of leaves for the grown tree.   |

|                |  |
|----------------|--|
| cp             | Complexity parameter, minimum loss decrease to split a node. A split is only performed if the loss decrease is larger than $cp * initial\_loss$ , where <code>initial_loss</code> is the loss of the initial estimate using only a stump.  |
| min_sample     | Minimum number of observations per leaf. A split is only performed if both resulting leaves have at least <code>min_sample</code> observations.  |
| mtry           | Number of randomly selected covariates to consider for a split, if NULL all covariates are available for each split.   |
| fast           | If TRUE, only the optimal splits in the new leaves are evaluated and the previously optimal splits and their potential loss-decrease are reused. If FALSE all possible splits in all the leaves are reevaluated after every split.   |
| Q_type         | Type of deconfounding, one of 'trim', 'pca', 'no_deconfounding'. 'trim' corresponds to the Trim transform (Ćevič et al. 2020) as implemented in the Doubly debiased lasso (Guo et al. 2022), 'pca' to the PCA transformation (Paul et al. 2008). See <a href="#">get_Q</a> .                       |
| trim_quantile  | Quantile for Trim transform, only needed for trim and DDL_trim, see <a href="#">get_Q</a> .  |
| q_hat          | Assumed confounding dimension, only needed for pca, see <a href="#">get_Q</a> .  |
| Qf             | Spectral transformation, if NULL it is internally estimated using <a href="#">get_Q</a> .  |
| A              | Numerical Anchor of class matrix. See <a href="#">get_W</a> .  |
| gamma          | Strength of distributional robustness, $\gamma \in [0, \infty]$ . See <a href="#">get_W</a> .  |
| gpu            | If TRUE, the calculations are performed on the GPU. If it is properly set up.  |
| mem_size       | Amount of split candidates that can be evaluated at once. This is a trade-off between memory and speed can be decreased if either the memory is not sufficient or the gpu is too small.  |
| max_candidates | Maximum number of split points that are proposed at each node for each covariate.  |
| nfolds         | Number of folds for cross-validation. It is recommended to not use more than 5 folds if the number of covariates is larger than the number of observations. In this case the spectral transformation could differ too much if the validation data is substantially smaller than the training data. |
| cp_seq         | Sequence of complexity parameters cp to compare using cross-validation, if NULL a sequence from 0 to 0.6 with stepsize 0.002 is used.  |
| mc.cores       | Number of cores to use for parallel computation.   |
| Q_scale        | Should data be scaled to estimate the spectral transformation? Default is TRUE to not reduce the signal of high variance covariates, and we do not know of a scenario where this hurts.  |

### Value

A list containing

|          |  |
|----------|--|
| cp_min   | The optimal complexity parameter.  |
| cp_table | A table containing the complexity parameter, the mean and the standard deviation of the loss on the validation sets for the complexity parameters. If multiple complexity parameters result in the same loss, only the one with the largest complexity parameter is shown. |

**Author(s)**

Markus Ulmer

**References**

Ćevič D, Bühlmann P, Meinshausen N (2020). “Spectral Deconfounding via Perturbed Sparse Linear Models.” *J. Mach. Learn. Res.*, **21**(1). ISSN 1532-4435, <http://jmlr.org/papers/v21/19-545.html>.

Guo Z, Ćevič D, Bühlmann P (2022). “Doubly debiased lasso: High-dimensional inference under hidden confounding.” *The Annals of Statistics*, **50**(3). ISSN 0090-5364, [doi:10.1214/21AOS2152](https://doi.org/10.1214/21AOS2152).

Paul D, Bair E, Hastie T, Tibshirani R (2008). ““Preconditioning” for feature selection and regression in high-dimensional problems.” *The Annals of Statistics*, **36**(4). ISSN 0090-5364, [doi:10.1214/009053607000000578](https://doi.org/10.1214/009053607000000578).

**See Also**[SDTree prune](#). [SDTree regPath](#). [SDTree](#)**Examples**

```
set.seed(1)
n <- 50
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n, 0, 5)
cp <- cvSDTree(x = X, y = y, Q_type = 'no_deconfounding')
cp
```

---

|                   |                                 |
|-------------------|---------------------------------|
| fromList.SDForest | <i>SDForest fromList method</i> |
|-------------------|---------------------------------|

---

**Description**

Converts the trees in an SDForest object from class `list` to class `Node` (Glur 2023).

**Usage**

```
## S3 method for class 'SDForest'
fromList(object, ...)
```

**Arguments**

|                     |  |
|---------------------|--|
| <code>object</code> | an SDForest object with the trees in list format   |
| <code>...</code>    | Further arguments passed to or from other methods. |

**Value**

an SDForest object with the trees in Node format

**Author(s)**

Markus Ulmer

**References**

Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.

**See Also**

[fromList](#) [fromList.SDTree](#)

**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5, nTree = 2)
fromList(toList(model))
```

---

|                 |                               |
|-----------------|-------------------------------|
| fromList.SDTree | <i>SDTree fromList method</i> |
|-----------------|-------------------------------|

---

**Description**

Converts the tree in an SDTree object from class list to class Node (Glur 2023).

**Usage**

```
## S3 method for class 'SDTree'
fromList(object, ...)
```

**Arguments**

|        |  |
|--------|--|
| object | an SDTree object with the tree in list format      |
| ...    | Further arguments passed to or from other methods. |

**Value**

an SDTree object with the tree in Node format

**Author(s)**

Markus Ulmer



**References**

Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.

**See Also**

[toList](#)

**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
fromList(toList(model))
```

---

f\_four

*Function of x on a fourier basis*


---

**Description**

Function of x on a fourier basis with a subset of covariates having a causal effect on Y using the parameters beta. The function is given by:

$$f(x_i) = \sum_{j=1}^p 1_{j \in js} \sum_{k=1}^K (\beta_{j,k}^{(1)} \cos(0.2kx_j) + \beta_{j,k}^{(2)} \sin(0.2kx_j))$$

**Usage**

```
f_four(x, beta, js)
```

**Arguments**

|      |  |
|------|--|
| x    | a vector of covariates                     |
| beta | the parameter vector for the function f(X) |
| js   | the indices of the causal covariates in X  |

**Value**

the value of the function f(x)

**Author(s)**

Markus Ulmer

**See Also**

[simulate\\_data\\_nonlinear](#)

**Examples**

```
set.seed(42)
# simulation of confounded data
sim_data <- simulate_data_nonlinear(q = 2, p = 150, n = 100, m = 2)
X <- sim_data$X
j <- sim_data$j[1]
apply(X, 1, function(x) f_four(x, sim_data$beta, j))
```

---

get\_cp\_seq.SDForest    *Get the sequence of complexity parameters of an SDForest*

---

**Description**

This function extracts the sequence of complexity parameters of an SDForest that result in changes of the SDForest if pruned. Only cp values that differ in the first three digits after the decimal point are returned.

**Usage**

```
## S3 method for class 'SDForest'
get_cp_seq(object, ...)
```

**Arguments**

object            an SDForest object  
...                Further arguments passed to or from other methods.

**Value**

A sequence of complexity parameters

**Author(s)**

Markus Ulmer

**See Also**

[regPath](#) [stabilitySelection](#) [get\\_cp\\_seq.SDTree](#)

**Examples**

```

set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', cp = 0, nTree = 2)
get_cp_seq(model)

```

---

get\_cp\_seq.SDTree      *Get the sequence of complexity parameters of an SDTree*

---

**Description**

This function extracts the sequence of complexity parameters of an SDTree that result in changes of the tree structure if pruned. Only cp values that differ in the first three digits after the decimal point are returned.

**Usage**

```

## S3 method for class 'SDTree'
get_cp_seq(object, ...)

```

**Arguments**

|        |  |
|--------|--|
| object | an SDTree object                                   |
| ...    | Further arguments passed to or from other methods. |

**Value**

A sequence of complexity parameters

**Author(s)**

Markus Ulmer

**See Also**

[regPath](#) [stabilitySelection](#)

**Examples**

```

set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0)
get_cp_seq(model)

```

---

 get\_Q

*Estimation of spectral transformation*


---

### Description

Estimates the spectral transformation  $Q$  for spectral deconfounding by shrinking the leading singular values of the covariates.

### Usage

```
get_Q(X, type, trim_quantile = 0.5, q_hat = 0, gpu = FALSE, scaling = TRUE)
```

### Arguments

|                            |   |
|----------------------------|---|
| <code>X</code>             | Numerical covariates of class matrix.   |
| <code>type</code>          | Type of deconfounding, one of 'trim', 'pca', 'no_deconfounding'. 'trim' corresponds to the Trim transform (Ćevic et al. 2020) as implemented in the Doubly debiased lasso (Guo et al. 2022), 'pca' to the PCA transformation (Paul et al. 2008) and 'no_deconfounding' to the Identity. |
| <code>trim_quantile</code> | Quantile for Trim transform, only needed for trim.  |
| <code>q_hat</code>         | Assumed confounding dimension, only needed for pca.   |
| <code>gpu</code>           | If TRUE, the calculations are performed on the GPU. If it is properly set up.   |
| <code>scaling</code>       | Whether $X$ should be scaled before calculating the spectral transformation.  |

### Value

$Q$  of class matrix, the spectral transformation matrix.

### Author(s)

Markus Ulmer

### References

Ćevic D, Bühlmann P, Meinshausen N (2020). "Spectral Deconfounding via Perturbed Sparse Linear Models." *J. Mach. Learn. Res.*, **21**(1). ISSN 1532-4435, <http://jmlr.org/papers/v21/19-545.html>.

Guo Z, Ćevic D, Bühlmann P (2022). "Doubly debiased lasso: High-dimensional inference under hidden confounding." *The Annals of Statistics*, **50**(3). ISSN 0090-5364, [doi:10.1214/21AOS2152](https://doi.org/10.1214/21AOS2152).

Paul D, Bair E, Hastie T, Tibshirani R (2008). "Preconditioning" for feature selection and regression in high-dimensional problems." *The Annals of Statistics*, **36**(4). ISSN 0090-5364, [doi:10.1214/009053607000000578](https://doi.org/10.1214/009053607000000578).

**Examples**

```
set.seed(1)
X <- matrix(rnorm(50 * 20), nrow = 50)
Q_trim <- get_Q(X, 'trim')
Q_pca <- get_Q(X, 'pca', q_hat = 5)
Q_plain <- get_Q(X, 'no_deconfounding')
```

get\_W

*Estimation of anchor transformation***Description**

Estimates the anchor transformation for the Anchor-Objective. The anchor transformation is  $W = I - (1 - \sqrt{\gamma})\Pi_A$ , where  $\Pi_A = A(A^T A)^{-1}A^T$ . For  $\gamma = 1$  this is just the identity. For  $\gamma = 0$  this corresponds to residuals after orthogonal projecting onto A. For large  $\gamma$  this is close to the orthogonal projection onto A, scaled by  $\gamma$ . The estimator  $\operatorname{argmin}_f \|W(Y - f(X))\|^2$  corresponds to the Anchor-Regression Estimator (Rothenhäusler et al. 2021), (Bühlmann 2020).

**Usage**

```
get_W(A, gamma, intercept = FALSE, gpu = FALSE)
```

**Arguments**

|           |   |
|-----------|---|
| A         | Numerical Anchor of class matrix.   |
| gamma     | Strength of distributional robustness, $\gamma \in [0, \infty]$ .             |
| intercept | Logical, whether to include an intercept in the anchor.                       |
| gpu       | If TRUE, the calculations are performed on the GPU. If it is properly set up. |

**Value**

W of class matrix, the anchor transformation matrix.

**Author(s)**

Markus Ulmer

**References**

Bühlmann P (2020). “Invariance, Causality and Robustness.” *Statistical Science*, **35**(3). ISSN 0883-4237, doi:10.1214/19STS721.

Rothenhäusler D, Meinshausen N, Bühlmann P, Peters J (2021). “Anchor Regression: Heterogeneous Data Meet Causality.” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **83**(2), 215–246. ISSN 1369-7412, doi:10.1111/rssb.12398.

## Examples

```
set.seed(1)
n <- 50
X <- matrix(rnorm(n * 1), nrow = n)
Y <- 3 * X + rnorm(n)
W <- get_W(X, gamma = 0)
resid <- W %*% Y
```

---

mergeForest

*Merge two forests*

---

## Description

This function merges two forests. The trees are combined and the variable importance is calculated as a weighted average of the two forests. If the forests are trained on the same data, the predictions and oob\_predictions are combined as well.

## Usage

```
mergeForest(fit1, fit2)
```

## Arguments

|      |                        |
|------|------------------------|
| fit1 | first SDForest object  |
| fit2 | second SDForest object |

## Value

```
merged SDForest object set.seed(1) n <- 50 X <- matrix(rnorm(n * 5), nrow = n) y <- sign(X[, 1])
* 3 + rnorm(n) fit1 <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', nTree = 5, cp = 0.5)
fit2 <- SDForest(x = X, y = y, nTree = 5, cp = 0.5) mergeForest(fit1, fit2)
```

## Author(s)

Markus Ulmer

---

|                |                           |
|----------------|---------------------------|
| partDependence | <i>Partial dependence</i> |
|----------------|---------------------------|

---

### Description

This function calculates the partial dependence of a model on a single variable. For that predictions are made for all observations in the dataset while varying the value of the variable of interest. The overall partial effect is the average of all predictions. (Friedman 2001)

### Usage

```
partDependence(object, j, X = NULL, subSample = NULL, mc.cores = 1)
```

### Arguments

|           |  |
|-----------|--|
| object    | A model object that has a predict method that takes newdata as argument and returns predictions.   |
| j         | The variable for which the partial dependence should be calculated. Either the column index of the variable in the dataset or the name of the variable.  |
| X         | The dataset on which the partial dependence should be calculated. Should contain the same variables as the dataset used to train the model. If NULL, tries to extract the dataset from the model object. |
| subSample | Number of samples to draw from the original data for the empirical partial dependence. If NULL, all the observations are used.   |
| mc.cores  | Number of cores to use for parallel computation. Parallel computing is only supported for unix.  |

### Value

An object of class `partDependence` containing

|            |  |
|------------|--|
| preds_mean | The average prediction for each value of the variable of interest.               |
| x_seq      | The sequence of values for the variable of interest.                             |
| preds      | The predictions for each value of the variable of interest for each observation. |
| j          | The name of the variable of interest.  |
| xj         | The values of the variable of interest in the dataset.                           |

### Author(s)

Markus Ulmer

### References

Friedman JH (2001). "Greedy Function Approximation: A Gradient Boosting Machine." *The Annals of Statistics*, **29**(5), 1189–1232. ISSN 00905364, <http://www.jstor.org/stable/2699986>.

**See Also**

[SDForest](#), [SDTree](#)

**Examples**

```
set.seed(1)
x <- rnorm(100)
y <- sign(x) * 3 + rnorm(100)
model <- SDTree(x = x, y = y, Q_type = 'no_deconfounding')
pd <- partDependence(model, 1, X = x, subSample = 10)
plot(pd)
```

---

plot.partDependence    *Plot partial dependence*

---

**Description**

This function plots the partial dependence of a model on a single variable.

**Usage**

```
## S3 method for class 'partDependence'
plot(x, n_examples = 19, ...)
```

**Arguments**

|            |  |
|------------|--|
| x          | An object of class partDependence returned by <a href="#">partDependence</a> . |
| n_examples | Number of examples to plot in addition to the average prediction.              |
| ...        | Further arguments passed to or from other methods.                             |

**Value**

A ggplot object.

**Author(s)**

Markus Ulmer

**See Also**

[partDependence](#) set.seed(1) x <- rnorm(10) y <- sign(x) \* 3 + rnorm(10) model <- SDTree(x = x, y = y, Q\_type = 'no\_deconfounding', cp = 0.5) pd <- partDependence(model, 1, X = x) plot(pd)



---

|            |   |
|------------|---|
| plot.paths | <i>Visualize the paths of an SDTree or SDForest</i> |
|------------|---|

---

### Description

This function visualizes the variable importance of an SDTree or SDForest for different complexity parameters. Both the regularization path and the stability selection path can be visualized.

### Usage

```
## S3 method for class 'paths'  
plot(x, plotly = FALSE, selection = NULL, sqrt_scale = FALSE, ...)
```

### Arguments

|            |  |
|------------|--|
| x          | A paths object   |
| plotly     | If TRUE the plot is returned interactive using plotly. Might be slow for large data.   |
| selection  | A vector of indices of the covariates to be plotted. Can be used to plot only a subset of the covariates in case of many covariates. |
| sqrt_scale | If TRUE the y-axis is on a square root scale.  |
| ...        | Further arguments passed to or from other methods.   |

### Value

A ggplot object with the variable importance for different regularization. If the path object includes a cp\_min value, a black dashed line is added to indicate the out-of-bag optimal variable selection.

### Author(s)

Markus Ulmer

### See Also

[regPath stabilitySelection](#)

### Examples

```
set.seed(1)  
n <- 10  
X <- matrix(rnorm(n * 5), nrow = n)  
y <- sign(X[, 1]) * 3 + sign(X[, 2]) + rnorm(n)  
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)  
paths <- regPath(model)  
plot(paths)
```

```
plot(paths, plotly = TRUE)
```

---

```
plot.SDTree
```

```
Plot SDTree
```

---

### Description

Plot the SDTree.

### Usage

```
## S3 method for class 'SDTree'
plot(x, ...)
```

### Arguments

`x` Fitted object of class SDTree.  
`...` Further arguments for DiagrammeR::render\_graph()

### Value

graph object from DiagrammeR::render\_graph()

### Author(s)

Markus Ulmer

### See Also

[SDTree](#) `set.seed(1) n <- 10 X <- matrix(rnorm(n * 5), nrow = n) y <- sign(X[, 1]) * 3 + rnorm(n)`  
`model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5) plot(model)`

---

```
plotOOB
```

```
Visualize the out-of-bag performance of an SDForest
```

---

### Description

This function visualizes the out-of-bag performance of an SDForest for different complexity parameters. Can be used to choose the optimal complexity parameter.

### Usage

```
plotOOB(object, sqrt_scale = FALSE)
```

**Arguments**

|            |   |
|------------|---|
| object     | A paths object with loss_path matrix with the out-of-bag performance for each complexity parameter. |
| sqrt_scale | If TRUE the x-axis is on a square root scale.   |

**Value**

A ggplot object

**Author(s)**

Markus Ulmer

**See Also**

[regPath.SDForest](#)

**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + sign(X[, 2]) + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
paths <- regPath(model)
plotOOB(paths)
```

---

predict.SDAM

*Predictions for SDAM*

---

**Description**

Predicts the response for new data using a fitted SDAM.

**Usage**

```
## S3 method for class 'SDAM'
predict(object, newdata, ...)
```

**Arguments**

|         |  |
|---------|--|
| object  | Fitted object of class SDAM.   |
| newdata | New test data of class data.frame containing the covariates for which to predict the response. |
| ...     | Further arguments passed to or from other methods.   |

**Value**

A vector of predictions for the new data.

**Author(s)**

Cyrill Scheidegger

**See Also**

[SDAM](#)

**Examples**

```
set.seed(1)
X <- matrix(rnorm(10 * 5), ncol = 5)
Y <- sin(X[, 1]) - X[, 2] + rnorm(10)
model <- SDAM(x = X, y = Y, Q_type = "trim", trim_quantile = 0.5, nfold = 2, n_K = 1)
predict(model, newdata = data.frame(X))
```

---

predict.SDForest

*Predictions for the SDForest*

---

**Description**

Predicts the response for new data using a fitted SDForest.

**Usage**

```
## S3 method for class 'SDForest'
predict(object, newdata, ...)
```

**Arguments**

|         |  |
|---------|--|
| object  | Fitted object of class SDForest.   |
| newdata | New test data of class data.frame containing the covariates for which to predict the response. |
| ...     | Further arguments passed to or from other methods.   |

**Value**

A vector of predictions for the new data.

**Author(s)**

Markus Ulmer

**See Also**[SDForest](#)**Examples**

```
set.seed(1)
n <- 50
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', nTree = 5, cp = 0.5)
predict(model, newdata = data.frame(X))
```

---

predict.SDTree

*Predictions for the SDTree*

---

**Description**

Predicts the response for new data using a fitted SDTree.

**Usage**

```
## S3 method for class 'SDTree'
predict(object, newdata, ...)
```

**Arguments**

|         |  |
|---------|--|
| object  | Fitted object of class SDTree.   |
| newdata | New test data of class data.frame containing the covariates for which to predict the response. |
| ...     | Further arguments passed to or from other methods.   |

**Value**

A vector of predictions for the new data.

**Author(s)**

Markus Ulmer

**See Also**[SDTree](#)

## Examples

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
predict(model, newdata = data.frame(X))
```

---

predictOOB

*Out-of-bag predictions for the SDForest*

---

## Description

Predicts the response for the training data using only the trees in the SDForest that were not trained on the observation.

## Usage

```
predictOOB(object, X = NULL)
```

## Arguments

|        |   |
|--------|---|
| object | Fitted object of class SDForest.  |
| X      | Covariates of the training data. If NULL, the data saved in the object is used. |

## Value

A vector of out-of-bag predictions for the training data. #' set.seed(1) n <- 50 X <- matrix(rnorm(n \* 5), nrow = n) y <- sign(X[, 1]) \* 3 + rnorm(n) model <- SDForest(x = X, y = y, Q\_type = 'no\_deconfounding', nTree = 5, cp = 0.5) predictOOB(model)

## Author(s)

Markus Ulmer

## See Also

[SDForest](#) [prune.SDForest](#) [plotOOB](#)

---

predict\_individual\_fj *Predictions of individual component functions for SDAM*

---

**Description**

Predicts the contribution of an individual component  $j$  using a fitted SDAM.

**Usage**

```
predict_individual_fj(object, j, newdata = NULL)
```

**Arguments**

|         |  |
|---------|--|
| object  | Fitted object of class SDAM.   |
| j       | Which component to evaluate.   |
| newdata | New test data of class data.frame containing the covariates for which to predict the response. |

**Value**

A vector of predictions for  $f_j$  evaluated at  $X_{jnew}$ .

**Author(s)**

Cyrill Scheidegger

**See Also**

[SDAM](#)

**Examples**

```
set.seed(1)
X <- matrix(rnorm(10 * 5), ncol = 5)
Y <- sin(X[, 1]) - X[, 2] + rnorm(10)
model <- SDAM(x = X, y = Y, Q_type = "trim", trim_quantile = 0.5, nfold = 2, n_K = 1)
predict_individual_fj(model, j = 1)
```

---

`print.partDependence` *Print partDependence*

---

**Description**

Print contents of the `partDependence`.

**Usage**

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

**Arguments**

`x`                    Fitted object of class `partDependence`.  
`...`                Further arguments passed to or from other methods.

**Value**

No return value, called for side effects

**Author(s)**

Markus Ulmer

**See Also**

[partDependence](#), [plot.partDependence](#)

**Examples**

```
set.seed(1)  
x <- rnorm(10)  
y <- sign(x) * 3 + rnorm(10)  
model <- SDTree(x = x, y = y, Q_type = 'no_deconfounding', cp = 0.5)  
pd <- partDependence(model, 1, X = x)  
print(pd)
```



---

|            |                   |
|------------|-------------------|
| print.SDAM | <i>Print SDAM</i> |
|------------|-------------------|

---

## Description

Print number of covariates and number of active covariates for SDAM object.

## Usage

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

## Arguments

|     |  |
|-----|--|
| x   | Fitted object of class SDAM.                       |
| ... | Further arguments passed to or from other methods. |

## Value

No return value, called for side effects

## Author(s)

Cyrill Scheidegger

## See Also

[SDAM](#)

## Examples

```
set.seed(1)  
X <- matrix(rnorm(10 * 5), ncol = 5)  
Y <- sin(X[, 1]) - X[, 2] + rnorm(10)  
model <- SDAM(x = X, y = Y, Q_type = "trim", trim_quantile = 0.5, nfold = 2, n_K = 1)  
print(model)
```

---

|                |                       |
|----------------|-----------------------|
| print.SDForest | <i>Print SDForest</i> |
|----------------|-----------------------|

---

### Description

Print contents of the SDForest.

### Usage

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

### Arguments

|     |  |
|-----|--|
| x   | Fitted object of class SDForest.                   |
| ... | Further arguments passed to or from other methods. |

### Value

No return value, called for side effects

### Author(s)

Markus Ulmer

### See Also

[SDForest](#)

### Examples

```
set.seed(1)  
n <- 50  
X <- matrix(rnorm(n * 5), nrow = n)  
y <- sign(X[, 1]) * 3 + rnorm(n)  
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', nTree = 5, cp = 0.5)  
print(model)
```

---

|              |                       |
|--------------|-----------------------|
| print.SDTree | <i>Print a SDTree</i> |
|--------------|-----------------------|

---

**Description**

Print contents of the SDTree.

**Usage**

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

**Arguments**

|     |  |
|-----|--|
| x   | Fitted object of class SDTree.                     |
| ... | Further arguments passed to or from other methods. |

**Value**

No return value, called for side effects

**Author(s)**

Markus Ulmer

**See Also**

[SDTree](#)

**Examples**

```
set.seed(1)  
n <- 10  
X <- matrix(rnorm(n * 5), nrow = n)  
y <- sign(X[, 1]) * 3 + rnorm(n)  
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)  
print(model)
```

---

|                |                          |
|----------------|--------------------------|
| prune.SDForest | <i>Prune an SDForest</i> |
|----------------|--------------------------|

---

### Description

Prunes all trees in the forest and re-calculates the out-of-bag predictions and performance measures. The training data is needed to calculate the out-of-bag statistics. Note that the forest is pruned in place. If you intend to keep the original forest, make a copy of it before pruning.

### Usage

```
## S3 method for class 'SDForest'
prune(object, cp, X = NULL, Y = NULL, Q = NULL, pred = TRUE, ...)
```

### Arguments

|        |  |
|--------|--|
| object | an SDForest object   |
| cp     | Complexity parameter, the higher the value the more nodes are pruned.  |
| X      | The training data, if NULL the data from the forest object is used.  |
| Y      | The training response variable, if NULL the data from the forest object is used.   |
| Q      | The transformation function, if NULL the data from the forest object is used.  |
| pred   | If TRUE the predictions are calculated, if FALSE only the out-of-bag statistics are calculated. This can set to FALSE to save computation time if only the out-of-bag statistics are needed. |
| ...    | Further arguments passed to or from other methods.   |

### Value

A pruned SDForest object

### Author(s)

Markus Ulmer

### See Also

[copy](#) [prune.SDTree](#) [regPath](#)

### Examples

```
set.seed(1)
X <- matrix(rnorm(10 * 20), nrow = 10)
Y <- rnorm(10)
fit <- SDForest(x = X, y = Y, nTree = 2)
pruned_fit <- prune(copy(fit), 0.2)
```

---

`prune.SDTree`*Prune an SDTree*

---

**Description**

Removes all nodes that did not improve the loss by more than `cp` times the initial loss. Either by themselves or by one of their successors. Note that the tree is pruned in place. If you intend to keep the original tree, make a copy of it before pruning.

**Usage**

```
## S3 method for class 'SDTree'  
prune(object, cp, ...)
```

**Arguments**

|                     |   |
|---------------------|---|
| <code>object</code> | an SDTree object  |
| <code>cp</code>     | Complexity parameter, the higher the value the more nodes are pruned. |
| <code>...</code>    | Further arguments passed to or from other methods.                    |

**Value**

A pruned SDTree object

**Author(s)**

Markus Ulmer

**See Also**

[copy](#)

**Examples**

```
set.seed(1)  
X <- matrix(rnorm(10 * 20), nrow = 10)  
Y <- rnorm(10)  
tree <- SDTree(x = X, y = Y)  
pruned_tree <- prune(copy(tree), 0.2)  
tree  
pruned_tree
```

---

regPath.SDForest      *Calculate the regularization path of an SDForest*

---

### Description

This function calculates the variable importance of an SDForest and the out-of-bag performance for different complexity parameters.

### Usage

```
## S3 method for class 'SDForest'
regPath(object, cp_seq = NULL, X = NULL, Y = NULL, Q = NULL, copy = TRUE, ...)
```

### Arguments

|        |  |
|--------|--|
| object | an SDForest object   |
| cp_seq | A sequence of complexity parameters. If NULL, the sequence is calculated automatically using only relevant values.   |
| X      | The training data, if NULL the data from the forest object is used.  |
| Y      | The training response variable, if NULL the data from the forest object is used.   |
| Q      | The transformation matrix, if NULL the data from the forest object is used.  |
| copy   | Whether the tree should be copied for the regularization path. If FALSE, the pruning is done in place and will change the SDForest. This might be reasonable, if the SDForest is to large to copy. |
| ...    | Further arguments passed to or from other methods.   |

### Value

An object of class paths containing

|             |   |
|-------------|---|
| cp          | The sequence of complexity parameters.                                  |
| varImp_path | A matrix with the variable importance for each complexity parameter.    |
| loss_path   | A matrix with the out-of-bag performance for each complexity parameter. |
| cp_min      | The complexity parameter with the lowest out-of-bag performance.        |
| type        | Path type   |

### Author(s)

Markus Ulmer

### See Also

[plot.paths](#) [plotOOB](#) [regPath.SDTree](#) [prune](#) [get\\_cp\\_seq](#) [SDForest](#)

**Examples**

```

set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + sign(X[, 2]) + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
paths <- regPath(model)
plotOOB(paths)
plot(paths)

plot(paths, plotly = TRUE)

```

---

regPath.SDTree

*Calculate the regularization path of an SDTree*


---

**Description**

This function calculates the variable importance of an SDTree for different complexity parameters.

**Usage**

```

## S3 method for class 'SDTree'
regPath(object, cp_seq = NULL, ...)

```

**Arguments**

|        |  |
|--------|--|
| object | an SDTree object   |
| cp_seq | A sequence of complexity parameters. If NULL, the sequence is calculated automatically using only relevant values. |
| ...    | Further arguments passed to or from other methods.   |

**Value**

An object of class paths containing

|             |  |
|-------------|--|
| cp          | The sequence of complexity parameters.                               |
| varImp_path | A matrix with the variable importance for each complexity parameter. |
| type        | Path type  |

**Author(s)**

Markus Ulmer

**See Also**

[plot.paths](#) [prune](#) [get\\_cp\\_seq](#) [SDTree](#)

## Examples

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + sign(X[, 2]) + rnorm(n)
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
paths <- regPath(model)
plot(paths)

plot(paths, plotly = TRUE)
```

---

SDAM

*Spectrally Deconfounded Additive Models*

---

## Description

Estimate high-dimensional additive models using spectral deconfounding (Scheidegger et al. 2025). The covariates are expanded into B-spline basis functions. A spectral transformation is used to remove bias arising from hidden confounding and a group lasso objective is minimized to enforce component-wise sparsity. Optimal number of basis functions per component and sparsity penalty are chosen by cross validation.

## Usage

```
SDAM(  
  formula = NULL,  
  data = NULL,  
  x = NULL,  
  y = NULL,  
  Q_type = "trim",  
  trim_quantile = 0.5,  
  q_hat = 0,  
  nfolds = 5,  
  cv_method = "1se",  
  n_K = 4,  
  n_lambda1 = 10,  
  n_lambda2 = 20,  
  Q_scale = TRUE,  
  ind_lin = NULL,  
  mc.cores = 1,  
  verbose = TRUE  
)
```



**Arguments**

|               |  |
|---------------|--|
| formula       | Object of class formula or describing the model to fit of the form $y \sim x_1 + x_2 + \dots$ where $y$ is a numeric response and $x_1, x_2, \dots$ are vectors of covariates. Interactions are not supported.   |
| data          | Training data of class data.frame containing the variables in the model.   |
| x             | Matrix of covariates, alternative to formula and data.   |
| y             | Vector of responses, alternative to formula and data.  |
| Q_type        | Type of deconfounding, one of 'trim', 'pca', 'no_deconfounding'. 'trim' corresponds to the Trim transform (Ćevič et al. 2020) as implemented in the Doubly debiased lasso (Guo et al. 2022), 'pca' to the PCA transformation (Paul et al. 2008). See <a href="#">get_Q</a> .         |
| trim_quantile | Quantile for Trim transform, only needed for trim, see <a href="#">get_Q</a> .   |
| q_hat         | Assumed confounding dimension, only needed for pca, see <a href="#">get_Q</a> .  |
| nfolds        | The number of folds for cross-validation. Default is 5.  |
| cv_method     | The method for selecting the regularization parameter during cross-validation. One of "min" (minimum cv-loss) and "1se" (one-standard-error rule) Default is "1se".  |
| n_K           | The number of candidate values for the number of basis functions for B-splines. Default is 4.  |
| n_lambda1     | The number of candidate values for the regularization parameter in the initial cross-validation step. Default is 10.   |
| n_lambda2     | The number of candidate values for the regularization parameter in the second stage of cross-validation (once the optimal number of basis function $K$ is decided, a second stage of cross-validation for the regularization parameter is performed on a finer grid). Default is 20. |
| Q_scale       | Should data be scaled to estimate the spectral transformation? Default is TRUE to not reduce the signal of high variance covariates.   |
| ind_lin       | A vector of indices specifying which covariates to model linearly (i.e. not expanded into basis function). Default is 'NULL'.  |
| mc.cores      | Number of cores to use for parallel processing, if <code>mc.cores &gt; 1</code> the cross validation is parallelized. Default is '1'. (only supported for unix)  |
| verbose       | If TRUE fitting information is shown.  |

**Value**

An object of class 'SDAM' containing the following elements:

|           |   |
|-----------|---|
| X         | The original design matrix.   |
| p         | The number of covariates in 'X'.  |
| intercept | The intercept term of the fitted model.   |
| K         | A vector of the number of basis functions for each covariate, where 1 corresponds to a linear term. The entries of the vector will mostly be the same, but some entries might be lower if the corresponding component of X contains only few unique values. |

|        |   |
|--------|---|
| breaks | A list of breakpoints used for the B-splines. Used to reconstruct the B-spline basis functions. |
| coefs  | A list of coefficients for the B-spline basis functions for each component.                     |
| active | A vector of active covariates that contribute to the model.                                     |

**Author(s)**

Cyrill Scheidegger

**References**

Ćevid D, Bühlmann P, Meinshausen N (2020). “Spectral Deconfounding via Perturbed Sparse Linear Models.” *J. Mach. Learn. Res.*, **21**(1). ISSN 1532-4435, <http://jmlr.org/papers/v21/19-545.html>.

Guo Z, Ćevid D, Bühlmann P (2022). “Doubly debiased lasso: High-dimensional inference under hidden confounding.” *The Annals of Statistics*, **50**(3). ISSN 0090-5364, [doi:10.1214/21AOS2152](https://doi.org/10.1214/21AOS2152).

Paul D, Bair E, Hastie T, Tibshirani R (2008). ““Preconditioning” for feature selection and regression in high-dimensional problems.” *The Annals of Statistics*, **36**(4). ISSN 0090-5364, [doi:10.1214/009053607000000578](https://doi.org/10.1214/009053607000000578).

Scheidegger C, Guo Z, Bühlmann P (2025). “Spectral Deconfounding for High-Dimensional Sparse Additive Models.” *ACM/IMS J. Data Sci.* [doi:10.1145/3711116](https://doi.org/10.1145/3711116).

**See Also**

[get\\_Q](#), [predict.SDAM](#), [varImp.SDAM](#), [predict\\_individual\\_fj](#), [partDependence](#)

**Examples**

```
set.seed(1)
X <- matrix(rnorm(10 * 5), ncol = 5)
Y <- sin(X[, 1]) - X[, 2] + rnorm(10)
model <- SDAM(x = X, y = Y, Q_type = "trim", trim_quantile = 0.5, nfold = 2, n_K = 1)
```

```
library(HDclassif)
data(wine)
names(wine) <- c("class", "alcohol", "malicAcid", "ash", "alcalinityAsh", "magnesium",
               "totPhenols", "flavanoids", "nonFlavPhenols", "proanthocyanins",
               "colIntens", "hue", "OD", "proline")
wine <- log(wine)
```

```
# estimate model
# do not use class in the model and restrict proline to be linear
model <- SDAM(alcohol ~ -class + ., wine, ind_lin = "proline", nfold = 3)
```

```
# extract variable importance
```

```

varImp(model)

# most important variable
mostImp <- names(which.max(varImp(model)))
mostImp

# predict for individual Xj
predJ <- predict_individual_fj(object = model, j = mostImp)
plot(wine[, mostImp], predJ,
     xlab = paste0("log ", mostImp), ylab = "log alcohol")

# partial dependence
plot(partDependence(model, mostImp))

# predict
predict(model, newdata = wine[42, ])

## alternative function call
mod_none <- SDAM(x = as.matrix(wine[1:10, -c(1, 2)]), y = wine$alcohol[1:10],
                 Q_type = "no_deconfounding", nolds = 2, n_K = 4,
                 n_lambda1 = 4, n_lambda2 = 8)

```

SDForest

*Spectrally Deconfounded Random Forests***Description**

Estimate regression Random Forest using spectral deconfounding. The spectrally deconfounded Random Forest (SDForest) combines SDTrees in the same way, as in the original Random Forest (Breiman 2001). The idea is to combine multiple regression trees into an ensemble in order to decrease variance and get a smooth function. Ensembles work best if the different models are independent of each other. To decorrelate the regression trees as much as possible from each other, we have two mechanisms. The first one is bagging (Breiman 1996), where we train each regression tree on an independent bootstrap sample of the observations, e.g., we draw a random sample of size  $n$  with replacement from the observations. The second mechanic to decrease the correlation is that only a random subset of the covariates is available for each split. Before each split, we sample  $m$  covariates from all the covariates and choose the one that reduces the loss the most only from those.

$$\widehat{f}(X) = \frac{1}{N_{tree}} \sum_{t=1}^{N_{tree}} SDTree_t(X)$$

**Usage**

```

SDForest(
  formula = NULL,
  data = NULL,
  x = NULL,

```

```

y = NULL,
nTree = 100,
cp = 0,
min_sample = 5,
mtry = NULL,
mc.cores = 1,
Q_type = "trim",
trim_quantile = 0.5,
q_hat = 0,
Qf = NULL,
A = NULL,
gamma = 7,
max_size = NULL,
gpu = FALSE,
return_data = TRUE,
mem_size = 1e+07,
leave_out_ind = NULL,
envs = NULL,
nTree_leave_out = NULL,
nTree_env = NULL,
max_candidates = 100,
Q_scale = TRUE,
verbose = TRUE
)

```

### Arguments

|            |  |
|------------|--|
| formula    | Object of class formula or describing the model to fit of the form $y \sim x_1 + x_2 + \dots$ where $y$ is a numeric response and $x_1, x_2, \dots$ are vectors of covariates. Interactions are not supported.   |
| data       | Training data of class data. frame containing the variables in the model.  |
| x          | Matrix of covariates, alternative to formula and data.   |
| y          | Vector of responses, alternative to formula and data.  |
| nTree      | Number of trees to grow.   |
| cp         | Complexity parameter, minimum loss decrease to split a node. A split is only performed if the loss decrease is larger than $cp * initial\_loss$ , where $initial\_loss$ is the loss of the initial estimate using only a stump.  |
| min_sample | Minimum number of observations per leaf. A split is only performed if both resulting leaves have at least <code>min_sample</code> observations.  |
| mtry       | Number of randomly selected covariates to consider for a split, if NULL half of the covariates are available for each split. $mtry = \lfloor \frac{p}{2} \rfloor$  |
| mc.cores   | Number of cores to use for parallel processing, if $mc.cores > 1$ the trees are estimated in parallel.   |
| Q_type     | Type of deconfounding, one of 'trim', 'pca', 'no_deconfounding'. 'trim' corresponds to the Trim transform (Čevič et al. 2020) as implemented in the Doubly debiased lasso (Guo et al. 2022), 'pca' to the PCA transformation (Paul et al. 2008). See <a href="#">get_Q</a> . |

|                 |   |
|-----------------|---|
| trim_quantile   | Quantile for Trim transform, only needed for trim, see <a href="#">get_Q</a> .  |
| q_hat           | Assumed confounding dimension, only needed for pca, see <a href="#">get_Q</a> .   |
| Qf              | Spectral transformation, if NULL it is internally estimated using <a href="#">get_Q</a> .   |
| A               | Numerical Anchor of class matrix. See <a href="#">get_W</a> .   |
| gamma           | Strength of distributional robustness, $\gamma \in [0, \infty]$ . See <a href="#">get_W</a> .   |
| max_size        | Maximum number of observations used for a bootstrap sample. If NULL n samples with replacement are drawn.   |
| gpu             | If TRUE, the calculations are performed on the GPU. If it is properly set up.   |
| return_data     | If TRUE, the training data is returned in the output. This is needed for <a href="#">prune.SDForest</a> , <a href="#">regPath.SDForest</a> , and for <a href="#">mergeForest</a> .      |
| mem_size        | Amount of split candidates that can be evaluated at once. This is a trade-off between memory and speed can be decreased if either the memory is not sufficient or the gpu is too small. |
| leave_out_ind   | Indices of observations that should not be used for training.   |
| envs            | Vector of environments of class factor which can be used for stratified tree fitting.   |
| nTree_leave_out | Number of trees that should be estimated while leaving one of the environments out. Results in number of environments times number of trees.  |
| nTree_env       | Number of trees that should be estimated for each environment. Results in number of environments times number of trees.   |
| max_candidates  | Maximum number of split points that are proposed at each node for each covariate.   |
| Q_scale         | Should data be scaled to estimate the spectral transformation? Default is TRUE to not reduce the signal of high variance covariates, and we do not know of a scenario where this hurts. |
| verbose         | If TRUE fitting information is shown.   |

### Value

Object of class SDForest containing:

|                |  |
|----------------|--|
| predictions    | Vector of predictions for each observation.  |
| forest         | List of SDTree objects.  |
| var_names      | Names of the covariates.   |
| oob_loss       | Out-of-bag loss. MSE   |
| oob_SDloss     | Out-of-bag loss using the spectral transformation.   |
| var_importance | Variable importance. The variable importance is calculated as the sum of the decrease in the loss function resulting from all splits that use a covariate for each tree. The mean of the variable importance of all trees results in the variable importance for the forest. |
| oob_ind        | List of indices of trees that did not contain the observation in the training set.   |

oob\_predictions  
Out-of-bag predictions.

If return\_data is TRUE the following are also returned:

X Matrix of covariates.  
Y Vector of responses.  
Q Spectral transformation.

If envs is provided the following are also returned:

envs Vector of environments.  
nTree\_env Number of trees for each environment.  
ooEnv\_ind List of indices of trees that did not contain the observation or the same environment in the training set for each observation.  
ooEnv\_loss Out-of-bag loss using only trees that did not contain the observation or the same environment.  
ooEnv\_SDloss Out-of-bag loss using the spectral transformation and only trees that did not contain the observation or the same environment.  
ooEnv\_predictions Out-of-bag predictions using only trees that did not contain the observation or the same environment.  
nTree\_leave\_out If environments are left out, the environment for each tree, that was left out.  
nTree\_env If environments are provided, the environment each tree is trained with.

### Author(s)

Markus Ulmer

### References

- Ćevic D, Bühlmann P, Meinshausen N (2020). “Spectral Deconfounding via Perturbed Sparse Linear Models.” *J. Mach. Learn. Res.*, **21**(1). ISSN 1532-4435, <http://jmlr.org/papers/v21/19-545.html>.
- Breiman L (1996). “Bagging predictors.” *Machine Learning*, **24**(2), 123–140. ISSN 0885-6125, [doi:10.1007/BF00058655](https://doi.org/10.1007/BF00058655).
- Breiman L (2001). “Random Forests.” *Machine Learning*, **45**(1), 5–32. ISSN 08856125, [doi:10.1023/A:1010933404324](https://doi.org/10.1023/A:1010933404324).
- Guo Z, Ćevic D, Bühlmann P (2022). “Doubly debiased lasso: High-dimensional inference under hidden confounding.” *The Annals of Statistics*, **50**(3). ISSN 0090-5364, [doi:10.1214/21AOS2152](https://doi.org/10.1214/21AOS2152).
- Paul D, Bair E, Hastie T, Tibshirani R (2008). ““Preconditioning” for feature selection and regression in high-dimensional problems.” *The Annals of Statistics*, **36**(4). ISSN 0090-5364, [doi:10.1214/009053607000000578](https://doi.org/10.1214/009053607000000578).

**See Also**

[get\\_Q](#), [get\\_W](#), [SDTree](#), [simulate\\_data\\_nonlinear](#), [regPath](#), [stabilitySelection](#), [prune](#), [partDependence](#)

**Examples**

```

set.seed(1)
n <- 50
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', nTree = 5, cp = 0.5)
predict(model, newdata = data.frame(X))

set.seed(42)
# simulation of confounded data
sim_data <- simulate_data_nonlinear(q = 2, p = 150, n = 100, m = 2)
X <- sim_data$X
Y <- sim_data$Y
train_data <- data.frame(X, Y)
# causal parents of y
sim_data$j

# comparison to classical random forest
fit_ranger <- ranger::ranger(Y ~ ., train_data, importance = 'impurity')

fit <- SDForest(x = X, y = Y, nTree = 10, Q_type = 'pca', q_hat = 2)
fit <- SDForest(Y ~ ., nTree = 10, train_data)
fit

# comparison of variable importance
imp_ranger <- fit_ranger$variable.importance
imp_sdf <- fit$var_importance
imp_col <- rep('black', length(imp_ranger))
imp_col[sim_data$j] <- 'red'

plot(imp_ranger, imp_sdf, col = imp_col, pch = 20,
     xlab = 'ranger', ylab = 'SDForest',
     main = 'Variable Importance')

# check regularization path of variable importance
path <- regPath(fit)
# out of bag error for different regularization
plotOOB(path)
plot(path)

# detection of causal parent using stability selection
stablePath <- stabilitySelection(fit)
plot(stablePath)

# pruning of forest according to optimal out-of-bag performance
fit <- prune(fit, cp = path$cp_min)

```

```
# partial functional dependence of y on the most important covariate
most_imp <- which.max(fit$var_importance)
dep <- partDependence(fit, most_imp)
plot(dep, n_examples = 100)
```

---

SDTree

*Spectrally Deconfounded Tree*


---

## Description

Estimates a regression tree using spectral deconfounding. A regression tree is part of the function class of step functions  $f(X) = \sum_{m=1}^M 1_{\{X \in R_m\}} c_m$ , where  $(R_m)$  with  $m = 1, \dots, M$  are regions dividing the space of  $\mathbb{R}^p$  into  $M$  rectangular parts. Each region has response level  $c_m \in \mathbb{R}$ . For the training data, we can write the step function as  $f(\mathbf{X}) = \mathcal{P}c$  where  $\mathcal{P} \in \{0, 1\}^{n \times M}$  is an indicator matrix encoding to which region an observation belongs and  $c \in \mathbb{R}^M$  is a vector containing the levels corresponding to the different regions. This function then minimizes

$$(\hat{\mathcal{P}}, \hat{c}) = \operatorname{argmin}_{\mathcal{P}' \in \{0, 1\}^{n \times M}, c' \in \mathbb{R}^M} \frac{\|Q(\mathbf{Y} - \mathcal{P}'c')\|_2^2}{n}$$

We find  $\hat{\mathcal{P}}$  by using the tree structure and repeated splitting of the leaves, similar to the original cart algorithm (Breiman et al. 2017). Since comparing all possibilities for  $\mathcal{P}$  is impossible, we let a tree grow greedily. Given the current tree, we iterate over all leaves and all possible splits. We choose the one that reduces the spectral loss the most and estimate after each split all the leave estimates  $\hat{c} = \operatorname{argmin}_{c' \in \mathbb{R}^M} \frac{\|Q\mathbf{Y} - Q\mathcal{P}'c'\|_2^2}{n}$  which is just a linear regression problem. This is repeated until the loss decreases less than a minimum loss decrease after a split. The minimum loss decrease equals a cost-complexity parameter  $cp$  times the initial loss when only an overall mean is estimated. The cost-complexity parameter  $cp$  controls the complexity of a regression tree and acts as a regularization parameter.

## Usage

```
SDTree(
  formula = NULL,
  data = NULL,
  x = NULL,
  y = NULL,
  max_leaves = NULL,
  cp = 0.01,
  min_sample = 5,
  mtry = NULL,
  fast = TRUE,
  Q_type = "trim",
  trim_quantile = 0.5,
  q_hat = 0,
  Qf = NULL,
  A = NULL,
```



```

    gamma = 0.5,
    gpu = FALSE,
    mem_size = 1e+07,
    max_candidates = 100,
    Q_scale = TRUE
  )

```

## Arguments

|                |  |
|----------------|--|
| formula        | Object of class formula or describing the model to fit of the form $y \sim x_1 + x_2 + \dots$ where $y$ is a numeric response and $x_1, x_2, \dots$ are vectors of covariates. Interactions are not supported.   |
| data           | Training data of class data.frame containing the variables in the model.   |
| x              | Matrix of covariates, alternative to formula and data.   |
| y              | Vector of responses, alternative to formula and data.  |
| max_leaves     | Maximum number of leaves for the grown tree.   |
| cp             | Complexity parameter, minimum loss decrease to split a node. A split is only performed if the loss decrease is larger than $cp * initial\_loss$ , where $initial\_loss$ is the loss of the initial estimate using only a stump.  |
| min_sample     | Minimum number of observations per leaf. A split is only performed if both resulting leaves have at least $min\_sample$ observations.  |
| mtry           | Number of randomly selected covariates to consider for a split, if NULL all covariates are available for each split.   |
| fast           | If TRUE, only the optimal splits in the new leaves are evaluated and the previously optimal splits and their potential loss-decrease are reused. If FALSE all possible splits in all the leaves are reevaluated after every split.   |
| Q_type         | Type of deconfounding, one of 'trim', 'pca', 'no_deconfounding'. 'trim' corresponds to the Trim transform (Čevid et al. 2020) as implemented in the Doubly debiased lasso (Guo et al. 2022), 'pca' to the PCA transformation (Paul et al. 2008). See <a href="#">get_Q</a> . |
| trim_quantile  | Quantile for Trim transform, only needed for trim, see <a href="#">get_Q</a> .   |
| q_hat          | Assumed confounding dimension, only needed for pca, see <a href="#">get_Q</a> .  |
| Qf             | Spectral transformation, if NULL it is internally estimated using <a href="#">get_Q</a> .  |
| A              | Numerical Anchor of class matrix. See <a href="#">get_W</a> .  |
| gamma          | Strength of distributional robustness, $\gamma \in [0, \infty]$ . See <a href="#">get_W</a> .  |
| gpu            | If TRUE, the calculations are performed on the GPU. If it is properly set up.  |
| mem_size       | Amount of split candidates that can be evaluated at once. This is a trade-off between memory and speed can be decreased if either the memory is not sufficient or the gpu is to small.   |
| max_candidates | Maximum number of split points that are proposed at each node for each covariate.  |
| Q_scale        | Should data be scaled to estimate the spectral transformation? Default is TRUE to not reduce the signal of high variance covariates, and we do not know of a scenario where this hurts.  |

**Value**

Object of class SDTree containing

|                |   |
|----------------|---|
| predictions    | Predictions for the training set.   |
| tree           | The estimated tree of class Node from (Glur 2023). The tree contains the information about all the splits and the resulting estimates.  |
| var_names      | Names of the covariates in the training data.   |
| var_importance | Variable importance of the covariates. The variable importance is calculated as the sum of the decrease in the loss function resulting from all splits that use this covariate. |

**Author(s)**

Markus Ulmer

**References**

- Ćevic D, Bühlmann P, Meinshausen N (2020). “Spectral Deconfounding via Perturbed Sparse Linear Models.” *J. Mach. Learn. Res.*, **21**(1). ISSN 1532-4435, <http://jmlr.org/papers/v21/19-545.html>.
- Breiman L, Friedman JH, Olshen RA, Stone CJ (2017). *Classification And Regression Trees*. Routledge. ISBN 9781315139470, [doi:10.1201/9781315139470](https://doi.org/10.1201/9781315139470).
- Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.
- Guo Z, Ćevic D, Bühlmann P (2022). “Doubly debiased lasso: High-dimensional inference under hidden confounding.” *The Annals of Statistics*, **50**(3). ISSN 0090-5364, [doi:10.1214/21AOS2152](https://doi.org/10.1214/21AOS2152).
- Paul D, Bair E, Hastie T, Tibshirani R (2008). ““Preconditioning” for feature selection and regression in high-dimensional problems.” *The Annals of Statistics*, **36**(4). ISSN 0090-5364, [doi:10.1214/009053607000000578](https://doi.org/10.1214/009053607000000578).

**See Also**

[simulate\\_data\\_nonlinear](#), [regPath.SDTree](#), [prune.SDTree](#), [partDependence](#)

**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDTree(x = X, y = y, cp = 0.5)
```

```
set.seed(42)
# simulation of confounded data
```

```

sim_data <- simulate_data_step(q = 2, p = 15, n = 100, m = 2)
X <- sim_data$X
Y <- sim_data$Y
train_data <- data.frame(X, Y)
# causal parents of y
sim_data$j

tree_plain_cv <- cvSDTree(Y ~ ., train_data, Q_type = "no_deconfounding")
tree_plain <- SDTree(Y ~ ., train_data, Q_type = "no_deconfounding", cp = 0)

tree_causal_cv <- cvSDTree(Y ~ ., train_data)
tree_causal <- SDTree(y = Y, x = X, cp = 0)

# check regularization path of variable importance
path <- regPath(tree_causal)
plot(path)

tree_plain <- prune(tree_plain, cp = tree_plain_cv$cp_min)
tree_causal <- prune(tree_causal, cp = tree_causal_cv$cp_min)
plot(tree_causal)
plot(tree_plain)

```

---

simulate\_data\_nonlinear

*Simulate data with linear confounding and non-linear causal effect*

---

## Description

Simulation of data from a confounded non-linear model. The data generating process is given by:

$$Y = f(X) + \delta^T H + \nu$$

$$X = \Gamma^T H + E$$

where  $f(X)$  is a random function on the fourier basis with a subset of size  $m$  covariates  $X_j$  having a causal effect on  $Y$ .

$$f(x_i) = \sum_{j=1}^p 1_{j \in j_s} \sum_{k=1}^K (\beta_{j,k}^{(1)} \cos(0.2kx_j) + \beta_{j,k}^{(2)} \sin(0.2kx_j))$$

$E, \nu$  are random error terms and  $H \in \mathbb{R}^{n \times q}$  is a matrix of random confounding covariates.  $\Gamma \in \mathbb{R}^{q \times p}$  and  $\delta \in \mathbb{R}^q$  are random coefficient vectors. For the simulation, all the above parameters are drawn from a standard normal distribution, except for  $\nu$  which is drawn from a normal distribution with standard deviation 0.1. The parameters  $\beta$  are drawn from a uniform distribution between -1 and 1.

## Usage

```
simulate_data_nonlinear(q, p, n, m, K = 2, eff = NULL, fixEff = FALSE)
```

**Arguments**

|        |   |
|--------|---|
| q      | number of confounding covariates in H   |
| p      | number of covariates in X   |
| n      | number of observations  |
| m      | number of covariates with a causal effect on Y  |
| K      | number of fourier basis functions $K \in \mathbb{N}$ , e.g. complexity of causal function   |
| eff    | the number of affected covariates in X by the confounding, if NULL all covariates are affected  |
| fixEff | if eff is smaller than p: If fixEff = TRUE, the causal parents are always affected by confounding if fixEff = FALSE, affected covariates are chosen completely at random. |

**Value**

a list containing the simulated data:

|      |   |
|------|---|
| X    | a matrix of covariates  |
| Y    | a vector of responses   |
| f_X  | a vector of the true function $f(X)$                                      |
| j    | the indices of the causal covariates in X                                 |
| beta | the parameter vector for the function $f(X)$ , see <a href="#">f_four</a> |
| H    | the matrix of confounding covariates                                      |

**Author(s)**

Markus Ulmer

**See Also**

[f\\_four](#)

**Examples**

```
set.seed(42)
# simulation of confounded data
sim_data <- simulate_data_nonlinear(q = 2, p = 150, n = 100, m = 2)
X <- sim_data$X
Y <- sim_data$Y
```

---

simulate\_data\_step     *Simulate data with linear confounding and causal effect following a step-function*

---

### Description

Simulation of data from a confounded non-linear model. Where the non-linear function is a random regression tree. The data generating process is given by:

$$Y = f(X) + \delta^T H + \nu$$

$$X = \Gamma^T H + E$$

where  $f(X)$  is a random regression tree with  $m$  random splits of the data. Resulting in a random step-function with  $m + 1$  levels, i.e. leaf-levels.

$$f(x_i) = \sum_{k=1}^K 1_{\{x_i \in R_k\}} c_k$$

$E, \nu$  are random error terms and  $H \in \mathbb{R}^{n \times q}$  is a matrix of random confounding covariates.  $\Gamma \in \mathbb{R}^{q \times p}$  and  $\delta \in \mathbb{R}^q$  are random coefficient vectors. For the simulation, all the above parameters are drawn from a standard normal distribution, except for  $\delta$  which is drawn from a normal distribution with standard deviation 10. The leaf levels  $c_k$  are drawn from a uniform distribution between -50 and 50.

### Usage

```
simulate_data_step(q, p, n, m, make_tree = FALSE)
```

### Arguments

|           |  |
|-----------|--|
| q         | number of confounding covariates in H                  |
| p         | number of covariates in X                              |
| n         | number of observations                                 |
| m         | number of covariates with a causal effect on Y         |
| make_tree | Whether the random regression tree should be returned. |

### Value

a list containing the simulated data:

|      |   |
|------|---|
| X    | a matrix of covariates  |
| Y    | a vector of responses   |
| f_X  | a vector of the true function $f(X)$                                    |
| j    | the indices of the causal covariates in X                               |
| tree | If make_tree, the random regression tree of class Node from (Glur 2023) |

**Author(s)**

Markus Ulmer

**References**

Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.

**See Also**[simulate\\_data\\_nonlinear](#)**Examples**

```
set.seed(42)
# simulation of confounded data
sim_data <- simulate_data_step(q = 2, p = 15, n = 100, m = 2)
X <- sim_data$X
Y <- sim_data$Y
```

---

`stabilitySelection.SDForest`*Calculate the stability selection of an SDForest*

---

**Description**

This function calculates the stability selection of an SDForest (Meinshausen and Bühlmann 2010). Stability selection is calculated as the fraction of trees in the forest that select a variable for a split at each complexity parameter.

**Usage**

```
## S3 method for class 'SDForest'
stabilitySelection(object, cp_seq = NULL, ...)
```

**Arguments**

|                     |  |
|---------------------|--|
| <code>object</code> | an SDForest object   |
| <code>cp_seq</code> | A sequence of complexity parameters. If NULL, the sequence is calculated automatically using only relevant values. |
| <code>...</code>    | Further arguments passed to or from other methods.   |

**Value**

An object of class paths containing

|                          |  |
|--------------------------|--|
| <code>cp</code>          | The sequence of complexity parameters.                               |
| <code>varImp_path</code> | A matrix with the stability selection for each complexity parameter. |
| <code>type</code>        | Path type  |

**Author(s)**

Markus Ulmer

**References**

Meinshausen N, Bühlmann P (2010). “Stability Selection.” *Journal of the Royal Statistical Society Series B: Statistical Methodology*, **72**(4), 417–473. ISSN 1369-7412, doi:10.1111/j.1467-9868.2010.00740.x.

**See Also**[plot.paths](#) [regPath](#) [prune](#) [get\\_cp\\_seq](#) [SDForest](#)**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + sign(X[, 2]) + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', nTree = 2, cp = 0.5)
paths <- stabilitySelection(model)
plot(paths)

plot(paths, plotly = TRUE)
```

---

`toList.SDForest`*SDForest toList method*

---

**Description**

Converts the trees in an SDForest object from class `Node` (Glur 2023) to class `list`. This makes it substantially easier to save the forest to disk.

**Usage**

```
## S3 method for class 'SDForest'
toList(object, ...)
```

**Arguments**

`object` an SDForest object with the trees in Node format  
`...` Further arguments passed to or from other methods.

**Value**

an SDForest object with the trees in list format

**Author(s)**

Markus Ulmer

**References**

Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.

**See Also**[fromList](#) [toList.SDTree](#)**Examples**

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDForest(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5, nTree = 2)
toList(model)
```

---

`toList.SDTree`*SDTree toList method*

---

**Description**

Converts the tree in an SDTree object from class Node (Glur 2023) to class list. This makes it substantially easier to save the tree to disk.

**Usage**

```
## S3 method for class 'SDTree'
toList(object, ...)
```

**Arguments**

|                     |  |
|---------------------|--|
| <code>object</code> | an SDTree object with the tree in Node format      |
| <code>...</code>    | Further arguments passed to or from other methods. |

**Value**

an SDTree object with the tree in list format

**Author(s)**

Markus Ulmer



## References

Glur C (2023). “data.tree: General Purpose Hierarchical Data Structure.” <https://CRAN.R-project.org/package=data.tree>.

## See Also

[fromList](#)

## Examples

```
set.seed(1)
n <- 10
X <- matrix(rnorm(n * 5), nrow = n)
y <- sign(X[, 1]) * 3 + rnorm(n)
model <- SDTree(x = X, y = y, Q_type = 'no_deconfounding', cp = 0.5)
toList(model)
```

---

varImp.SDAM

*Extract Variable importance for SDAM*

---

## Description

This function extracts the variable importance of an SDAM. The variable importance is calculated as the empirical squared L2 norm of fj. The measure is not standardized.

## Usage

```
## S3 method for class 'SDAM'
varImp(object)
```

## Arguments

object            an SDAM object

## Value

A vector of variable importance

## Author(s)

Cyrill Scheidegger

## See Also

[SDAM](#)

## Examples

```
set.seed(1)
X <- matrix(rnorm(10 * 5), ncol = 5)
Y <- sin(X[, 1]) - X[, 2] + rnorm(10)
model <- SDAM(x = X, y = Y, Q_type = "trim", trim_quantile = 0.5, nfold = 2)
varImp(model)
```

---

|                 |   |
|-----------------|---|
| varImp.SDForest | <i>Extract variable importance of an SDForest</i> |
|-----------------|---|

---

## Description

This function extracts the variable importance of an SDForest. The variable importance is calculated as the sum of the decrease in the loss function resulting from all splits that use a covariate for each tree. The mean of the variable importance of all trees results in the variable importance for the forest.

## Usage

```
## S3 method for class 'SDForest'
varImp(object)
```

## Arguments

object            an SDForest object

## Value

A named vector of variable importance

## Author(s)

Markus Ulmer

## See Also

[varImp.SDTree](#) [SDForest](#)

## Examples

```
data(iris)
fit <- SDForest(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width,
               iris, nTree = 10, cp = 0.5)
varImp(fit)
```

---

|               |   |
|---------------|---|
| varImp.SDTree | <i>Extract variable importance of an SDTree</i> |
|---------------|---|

---

**Description**

This function extracts the variable importance of an SDTree. The variable importance is calculated as the sum of the decrease in the loss function resulting from all splits that use this covariate.

**Usage**

```
## S3 method for class 'SDTree'  
varImp(object)
```

**Arguments**

object            an SDTree object

**Value**

A named vector of variable importance

**Author(s)**

Markus Ulmer

**See Also**

[varImp.SDForest](#) [SDTree](#)

**Examples**

```
data(iris)  
tree <- SDTree(Sepal.Length ~ Sepal.Width + Petal.Length + Petal.Width, iris, cp = 0.5)  
varImp(tree)
```

# Index

copy, [28](#), [29](#)  
copy (copy.SDForest), [3](#)  
copy.SDForest, [3](#)  
copy.SDTree, [4](#)  
cvSDTree, [5](#)

f\_four, [9](#), [44](#)  
fromList, [8](#), [48](#), [49](#)  
fromList (fromList.SDForest), [7](#)  
fromList.SDForest, [7](#)  
fromList.SDTree, [8](#), [8](#)

get\_cp\_seq, [30](#), [31](#), [47](#)  
get\_cp\_seq (get\_cp\_seq.SDForest), [10](#)  
get\_cp\_seq.SDForest, [10](#)  
get\_cp\_seq.SDTree, [10](#), [11](#)  
get\_Q, [6](#), [12](#), [33](#), [34](#), [36](#), [37](#), [39](#), [41](#)  
get\_W, [6](#), [13](#), [37](#), [39](#), [41](#)

mergeForest, [14](#), [37](#)

partDependence, [15](#), [16](#), [24](#), [34](#), [39](#), [42](#)  
plot.partDependence, [16](#), [24](#)  
plot.paths, [17](#), [30](#), [31](#), [47](#)  
plot.SDTree, [18](#)  
plotOOB, [18](#), [22](#), [30](#)  
predict.SDAM, [19](#), [34](#)  
predict.SDForest, [20](#)  
predict.SDTree, [21](#)  
predict\_individual\_fj, [23](#), [34](#)  
predictOOB, [22](#)  
print.partDependence, [24](#)  
print.SDAM, [25](#)  
print.SDForest, [26](#)  
print.SDTree, [27](#)  
prune, [3](#), [4](#), [30](#), [31](#), [39](#), [47](#)  
prune (prune.SDForest), [28](#)  
prune.SDForest, [22](#), [28](#), [37](#)  
prune.SDTree, [7](#), [28](#), [29](#), [42](#)

regPath, [10](#), [11](#), [17](#), [28](#), [39](#), [47](#)  
regPath (regPath.SDForest), [30](#)  
regPath.SDForest, [19](#), [30](#), [37](#)  
regPath.SDTree, [7](#), [30](#), [31](#), [42](#)

SDAM, [20](#), [23](#), [25](#), [32](#), [49](#)  
SDForest, [16](#), [21](#), [22](#), [26](#), [30](#), [35](#), [47](#), [50](#)  
SDTree, [7](#), [16](#), [18](#), [21](#), [27](#), [31](#), [39](#), [40](#), [51](#)  
simulate\_data\_nonlinear, [10](#), [39](#), [42](#), [43](#),  
[46](#)  
simulate\_data\_step, [45](#)  
stabilitySelection, [10](#), [11](#), [17](#), [39](#)  
stabilitySelection  
(stabilitySelection.SDForest),  
[46](#)  
stabilitySelection.SDForest, [46](#)

toList, [9](#)  
toList (toList.SDForest), [47](#)  
toList.SDForest, [47](#)  
toList.SDTree, [48](#), [48](#)

varImp (varImp.SDForest), [50](#)  
varImp.SDAM, [34](#), [49](#)  
varImp.SDForest, [50](#), [51](#)  
varImp.SDTree, [50](#), [51](#)