

# Package ‘iteremoval’

January 5, 2022

**Title** Iteration removal method for feature selection

**Version** 1.15.1

**Author** Jiacheng Chuan [aut, cre]

**Maintainer** Jiacheng Chuan <jiacheng\_chuan@outlook.com>

**Description** The package provides a flexible algorithm to screen features of two distinct groups in consideration of overfitting and overall performance. It was originally tailored for methylation locus screening of NGS data, and it can also be used as a generic method for feature selection. Each step of the algorithm provides a default method for simple implementation, and the method can be replaced by a user defined function.

**Depends** R (>= 3.5.0), ggplot2 (>= 2.2.1)

**Imports** magrittr, graphics, utils, GenomicRanges, SummarizedExperiment

**URL** <https://github.com/cihga39871/iteremoval>

**BugReports** <https://github.com/cihga39871/iteremoval/issues>

**License** GPL-2

**Encoding** UTF-8

**LazyData** true

**RoxygenNote** 6.0.1

**biocViews** StatisticalMethod

**VignetteBuilder** knitr

**Suggests** testthat, knitr

**git\_url** <https://git.bioconductor.org/packages/iteremoval>

**git\_branch** master

**git\_last\_commit** ab15019

**git\_last\_commit\_date** 2021-11-20

**Date/Publication** 2022-01-05

## R topics documented:

feature_hist . . . . .	2
feature_prevalence . . . . .	3
feature_removal . . . . .	4
feature_screen . . . . .	6
funcOrExp . . . . .	7
ggiteration_trace . . . . .	7
score_combine_methods . . . . .	8
score_standardization_methods . . . . .	9
SummarizedData . . . . .	9
SWRG0 . . . . .	10
SWRG1 . . . . .	11
weight_methods . . . . .	11

<b>Index</b>	<b>13</b>
--------------	-----------

---

feature_hist	<i>Plot histogram of feature prevalence</i>
--------------	---

---

### Description

Compute the feature prevalence (present in different cutoffs) after removing the features of the first index iterations, and then plot the histogram of remaining features. It calls `feature_prevalence(..., hist.plot=TRUE)`.

### Usage

```
feature_hist(li, index)
```

### Arguments

li	the list result of <code>feature_removal</code> .
index	removing the features of the first index iterations. It allows a positive integer or a proper fraction. If improper fraction, it is regarded as <code>as.integer(index)</code> .

### Value

histogram

### Examples

```
g1 <- SWRG1; g0 <- SWRG0

result.complex <- feature_removal(g1, g0,
  cutoff1=0.95, cutoff0=0.925,
  offset=c(0.5, 1, 2))

# index is a proportion in 0-1
```

```
feature_hist(result.complex, 0.5)

# index is a positive integer
feature_hist(result.complex, 233)
```

---

feature_prevalence	<i>Feature prevalence</i>
--------------------	---------------------------

---

### Description

Compute the feature prevalence after removing the features of the first index iterations.

### Usage

```
feature_prevalence(li, index, hist.plot = TRUE)
```

### Arguments

li	the list result of feature_removal.
index	removing the features of the first index iterations. It allows a positive integer or a proper fraction. If improper fraction, it is regarded as <code>as.integer(index)</code> .
hist.plot	bool. A switch to plot the histogram of the remaining features.

### Value

Matrix

### Examples

```
g1 <- SWRG1; g0 <- SWRG0

result.complex <- feature_removal(g1, g0,
  cutoff1=0.95, cutoff0=0.925,
  offset=c(0.5, 1, 2))

# index is a proportion in 0-1
prevalence.result <- feature_prevalence(result.complex, 0.5, hist.plot=TRUE)

# index is a positive integer
prevalence.result <- feature_prevalence(result.complex, 233, hist.plot=TRUE)
```

---

feature_removal	<i>Stepwise feature removal method</i>
-----------------	--

---

### Description

This function screens features iteratively in consideration of limiting overfitting and overall performance.

### Usage

```
feature_removal(g1 = NULL, g0 = NULL, cutoff1, cutoff0, lt = ">",
  offset = 1, weight.method = reciprocal_colSums,
  scoreStandardization.method = min_max,
  scoreCombine.method = linear_combine, SE = NULL, g0.filter = NULL, ...)
```

### Arguments

<code>g1</code>	a dataframe with the row of feature, and the column of observation. Cells are numeric or bool. If NULL, input data should be param SE and <code>g0.filter</code> .
<code>g0</code>	a dataframe with the same row names as <code>g1</code> . Normally, the observations in <code>g0</code> are in the distinct group of <code>g1</code> . If NULL, input data should be param SE and <code>g0.filter</code> .
<code>cutoff1</code>	<code>g1</code> is converted to a dataframe filled with 1 or 0 by <code>cutoff1</code> and <code>lt</code> . The result is called <code>g1.signal</code> . For example, if <code>lt="&gt;"</code> , the result of the step is <code>g1.signal &lt;- g1 &gt; cutoff1</code> . If you do not want the conversion, let <code>lt="skip"</code> .
<code>cutoff0</code>	<code>g0</code> is converted to dataframes of 1 or 0 by <code>cutoff0</code> and <code>lt</code> . It has the same usage as <code>cutoff1</code> . Different <code>cutoff1</code> and <code>cutoff0</code> influence overfitting.
<code>lt</code>	An operator to compare <code>gx</code> and <code>cutoffx</code> . Default is <code>"&gt;"</code> . Other options include <code>"&gt;="</code> , <code>"&lt;="</code> , <code>"&lt;"</code> , etc. Additionally, <code>lt="skip"</code> skips the comparison and <code>cutoffx</code> will be ignored.
<code>offset</code>	a parameter in <code>scoreCombine.method</code> . It adjusts the score proportion of <code>g1</code> and <code>g2</code> . Besides, <code>offset</code> can be a number or a numeric vector. If it is a vector, the overall iteration is done for each offset respectively. See more in parameter <code>scoreCombine.method</code> .
<code>weight.method</code>	<code>gx.weight</code> , weight of <code>gx</code> , is computed using <code>weight.method</code> . The weight is for the observations/columns, not the features/rows. The default weight method is <code>reciprocal_colSums</code> , ie. $1 / (1 + \text{colSums}(gx.\text{signal}, \text{na.rm=T}))$ . You can specify your own function, and the first parameter of the function should be the exact word of <code>gx.signal</code> .
<code>scoreStandardization.method</code>	Default standardization method is Min-Max, ie. normalizing the vector to 0-1 range. You can specify your own function, and the first parameter of the function is the sum-up dataframe. See more in Details section.

<code>scoreCombine.method</code>	to combine the feature score vectors of <code>g1</code> and <code>g0</code> . This method must have three parameters in order, <code>g1.score.feature</code> , <code>g0.score.feature</code> , and <code>offset</code> . Default method is <code>linear_combine</code> . <code>offset</code> in the default method adjusts the proportion of <code>g1.score.feature</code> . Specifically, <code>g1.score.feature * offset + g0.score.feature</code> . Besides, <code>offset</code> can be a number or a vector. If it is a vector, the overall iteration is done for each <code>offset</code> respectively.
<code>SE</code>	a <code>SummarizedExperiment</code> object. If <code>NULL</code> , input data should be <code>g1</code> and <code>g0</code> .
<code>g0.filter</code>	a logical vector <code>g0.filter</code> to define <code>SE</code> 's columns that belong to <code>g0</code> . If <code>NULL</code> , input data should be param <code>g1</code> and <code>g0</code> .
<code>...</code>	Other parameter passed to method of expression class.

### Details

The method removes one feature/row in each iteration, and requires (A) two dataframes, `g1` and `g0`, with identical row names; OR (B) A `SummarizedExperiment` object `SE`, and a logical vector `g0.filter` to define `SE`'s columns that belong to `g0`. Normally, `g0` is the control set. `SE` will be divided to `g1` and `g0` automatically.

In each iteration, first, `g1` and `g0` are converted to dataframes of 1 or 0 by `cutoff1`, `cutoff0`, and `lt`. The converted dataframes are called `gx.signal`, and `x` stands for 1 and 0. If you do not want the conversion, let `lt="skip"`, and cutoffs will be ignored.

Second, `gx.weight`, weight of `gx`, is computed using `weight.method`. The weight is for the observations/columns, not the features/rows. The default weight method is `reciprocal_colSums`, ie.  $1 / (1 + \text{colSums}(gx.\text{signal}, na.rm=T))$ . You can specify your own function, and the first parameter of the function should be the exact word of `gx.signal`.

Third, `gx.score`, the score dataframe for observations and features, is computed. It is the result of dot product of `gx.signal` and `gx.weight`.

Then, Summing up `gx.score` by row, and the result is standardized with function `scoreStandardization.method`. Default standardization method is `Min-Max`, ie. normalizing the vector to 0-1 range. You can specify your own function, and the first parameter of the function is the sum-up dataframe.

After that, `gx.score.feature`, the feature scores of `gx` are calculated. Now using `scoreCombine.method` to combine the feature score vectors of `g1` and `g0`. This method must have three parameters in order, `g1.score.feature`, `g0.score.feature`, and `offset`. Default method is `linear_combine`. `offset` in the default method adjusts the proportion of `g1.score.feature`. Specifically, `g1.score.feature * offset + g0.score.feature`. Besides, `offset` can be a number or a vector. If it is a vector, the overall iteration is done for each `offset` respectively.

### Value

a list with names "offset", "removed.feature\_names", "removed.scores", and "max.scores".

### Other usages

```
feature_removal(g1, g0, cutoff1, cutoff0, lt = ">", offset = 1, weight.method = reciprocal_colSums,
scoreStandardization.method = min_max, scoreCombine.method = linear_combine, ...)
```

```
feature_removal(SE, g0.filter, cutoff1, cutoff0, lt = ">", offset = 1, weight.method = reciprocal_colSums,
scoreStandardization.method = min_max, scoreCombine.method = linear_combine, ...)
```

**Author(s)**

Jiacheng CHUAN

**Examples**

```
g1 <- SWRG1; g0 <- SWRG0
result.simple.A <- feature_removal(g1, g0, cutoff1=0.95, cutoff0=0.95)

result.simple.B <- feature_removal(SummarizedData, SummarizedData$Group==0,
  cutoff1=0.95, cutoff0=0.95)

result.complex <- feature_removal(g1, g0,
  cutoff1=0.95, cutoff0=0.925, lt=">",
  offset=c(0.5, 2),
  weight.method="reciprocal_colSums",
  scoreStandardization.method="min_max",
  scoreCombine.method="linear_combine")
```

---

feature_screen	<i>Screening feature using prevalence</i>
----------------	---

---

**Description**

Return the screened feature names.

**Usage**

```
feature_screen(features, prevalence)
```

**Arguments**

features	result of feature_prevalence(...)
prevalence	the prevalence cutoff of features. The features with prevalence less than prevalence are removed.

**Value**

Vector

**Examples**

```
g1 <- SWRG1; g0 <- SWRG0

result.complex <- feature_removal(g1, g0,
  cutoff1=0.95, cutoff0=0.925,
  offset=c(0.5, 1, 2))

prevalence.result <- feature_prevalence(result.complex, 233, hist.plot=TRUE)
```

```
feature.list <- feature_screen(prevalence.result, 3)
```

---

funcOrExp

*Using function or parsing expression for normal class*


---

### Description

Evaluating a function or expression by the type of method.

### Usage

```
funcOrExp(method, ...)
```

### Arguments

method	if the class of method is "character", regarding method as a function and evaluating THE_METHOD(...). If the class of method is "function", return the result of method(...). If the class of method is "expression", return the result of eval(method).
...	arguments passed onto method if the class of method is "character" or "function".

### Value

Evaluated result.

### Examples

```
funcOrExp(sample, 5)
funcOrExp('sample', 5)
funcOrExp(parse(text='sample(5)'))
```

---

ggiteration\_trace

*Iteration trace of removed scores*


---

### Description

plot the score of removed feature in each iteration.

### Usage

```
ggiteration_trace(li)
```

### Arguments

li	the list result of feature_removal.
----	-------------------------------------

**Value**

ggplot2 object.

**Examples**

```
g1 <- SWRG1; g0 <- SWRG0

result.complex <- feature_removal(g1, g0,
  cutoff1=0.95, cutoff0=0.925,
  offset=c(0.5, 1, 2))

# it is a ggplot2 object, so plus sign is available
ggiteration_trace(result.complex) + theme_bw()
```

---

score\_combine\_methods *Score combine methods*

---

**Description**

Score combine methods of function `feature_removal`. To combine the feature score vectors of `g1` and `g0`. The method used in `feature_removal` must have three parameters in order, `g1.score.feature`, `g0.score.feature`, and `offset`. See details in the help page of function `feature_removal`.

**Usage**

```
linear_combine(g1.score.feature, g0.score.feature, offset = 1)
```

**Arguments**

<code>g1.score.feature</code>	feature score vector for <code>g1</code> .
<code>g0.score.feature</code>	feature score vector for <code>g0</code> .
<code>offset</code>	adjusts the proportion of <code>g1.score.feature</code> . Default is 1.

**Value**

`g1.score.feature * offset + g0.score.feature`.

**Examples**

```
linear_combine(0.2, 0.3, 2)
linear_combine(1:2, 3:4, 1)
```



---

score\_standardization\_methods  
*Score standardization methods*

---

**Description**

Score standardization methods of function `feature.removal`.

**Usage**

```
min_max(x, na.rm = TRUE)
```

**Arguments**

<code>x</code>	The first parameter of score standardization method used in <code>feature.removal</code> is the sum-up dataframe. See details in the help page of function <code>feature.removal</code> .
<code>na.rm</code>	Bool. Remove NA or not.

**Value**

Numeric, normalized `x` to 0-1 range.

**Examples**

```
min_max(1:5)
min_max(c(1:5, NA), na.rm=TRUE)
```

---

SummarizedData      *Summarized data*

---

**Description**

`RangedSummarizedExperiment` object contains the probability data of the control and positive samples. It is the integration of `SWRG0` and `SWRG1`. `colData` in `SummarizedData` contains a column 'Group' indicating normal(0) from malignant(1). In assay, column represents samples, row represents genome ranges related to a disease.

**Usage**

```
SummarizedData
```

**Format**

```
RangedSummarizedExperiment.
```

**Details**

We identified 299 genomic regions related to the methylation status of a disease. Then, we sequenced the cfDNA of 44 subjects. 22 were malignant, while others were normal. We built a statistical model to compute the probability of the disease for each region and subject, and this dataframe contains the probabilities of 299 regions and 44 samples.

**See Also**

Other data: [SWRG0](#), [SWRG1](#)

---

SWRG0

*Group 0 data*

---

**Description**

Probability data of control samples. Column represents samples, row represents genome ranges related to a disease.

**Usage**

SWRG0

**Format**

A data frame with 22 variables: G0Sample1, G0Sample2, ..., G0Sample22.

**Details**

We identified 299 genomic regions related to the methylation status of a disease. Then, we sequenced the cfDNA of 44 subjects. 22 were malignant, while others were normal. We built a statistical model to compute the probability of the disease for each region and subject, and this dataframe contains the probabilities of 299 regions and 22 normal samples.

**See Also**

Other data: [SWRG1](#), [SummarizedData](#)

---

 SWRG1

*Group 1 data*


---

**Description**

Probability data of positive samples. Column represents samples, row represents genome ranges related to a disease.

**Usage**

```
SWRG1
```

**Format**

A data frame with 22 variables: G0Sample1, G0Sample2, ..., G0Sample22.

**Details**

We identified 299 genomic regions related to the methylation status of a disease. Then, we sequenced the cfDNA of 44 subjects. 22 were malignant, while others were normal. We built a statistical model to compute the probability of the disease for each region and subject, and this dataframe contains the probabilities of 299 regions and 22 malignant samples.

**See Also**

Other data: [SWRG0](#), [SummarizedData](#)

---

 weight\_methods

*Weight methods*


---

**Description**

Weight methods of function `feature_removal`.

**Usage**

```
reciprocal_colSums(gx.signal)
```

```
ones(gx.signal)
```

**Arguments**

`gx.signal` The first parameter of the weight method used in `feature_removal` must be the exact word `gx.signal`.

**Value**

```
1 / (1 + colSums(gx.singal, na.rm=T)).  
1.
```

**Examples**

```
gx.singal <- data.frame(x=1:5, t=2:6, k=c(3:6, NA))  
reciprocal_colSums(gx.singal)  
1 == ones(2)  
1 == ones(c(4,6))
```

# Index

## \* datasets

SummarizedData, [9](#)

SWRG0, [10](#)

SWRG1, [11](#)

feature\_hist, [2](#)

feature\_prevalence, [3](#)

feature\_removal, [4](#)

feature\_screen, [6](#)

funcOrExp, [7](#)

ggiteration\_trace, [7](#)

linear\_combine (score\_combine\_methods),  
[8](#)

min\_max  
(score\_standardization\_methods),  
[9](#)

ones (weight\_methods), [11](#)

reciprocal\_colSums (weight\_methods), [11](#)

score\_combine\_methods, [8](#)

score\_standardization\_methods, [9](#)

SummarizedData, [9](#), [10](#), [11](#)

SWRG0, [10](#), [10](#), [11](#)

SWRG1, [10](#), [11](#)

weight\_methods, [11](#)