Package 'findIPs'

January 19, 2026

Type Package

Title Influential Points Detection for Feature Rankings

Version 1.6.0

Date 2024-11-20

Description Feature rankings can be distorted by a single case in the context of high-dimensional data. The cases exerts abnormal influence on feature rankings are called influential points (IPs). The package aims at detecting IPs based on case deletion and quantifies their effects by measuring the rank changes (DOI:10.48550/arXiv.2303.10516). The package applies a novel rank comparing measure using the adaptive weights that stress the top-ranked important features and adjust the weights to ranking properties.

License GPL-3

URL https://github.com/ShuoStat/findIPs

BugReports https://github.com/ShuoStat/findIPs

Depends graphics, R (>= 4.4.0)

Imports Biobase, BiocParallel, parallel, stats, SummarizedExperiment, survival, utils

Suggests BiocStyle, knitr, rmarkdown, testthat

VignetteBuilder knitr

biocViews GeneExpression, DifferentialExpression, Regression, Survival

Encoding UTF-8

LazyData FALSE

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

git_url https://git.bioconductor.org/packages/findIPs

git_branch RELEASE_3_22

git_last_commit b043ffe

git_last_commit_date 2025-10-29

Repository Bioconductor 3.22

Date/Publication 2026-01-19

Author Shuo Wang [aut, cre] (ORCID: <https://orcid.org/0009-0000-0424-2160>), Junyan Lu [aut]

Maintainer Shuo Wang <wangsures@foxmail.com>

Contents

---

findIPs — *Function to detect influential points for feature rankings*

Description

findIPs employs two important functions: getdrop1ranks and sumRanks. getdrop1ranks can calculate the original feature ranking and leave-one-out feature rankings. The outputs are subsequently taken to sumRanks, which computes the overall rank changes for each observation, indicating their influence on feature rankings.

Usage

```
findIPs(
  X,
  y,
  fun,
  decreasing = FALSE,
  topN = 100,
  method = "adaptive",
  nCores = NULL
)
```

Arguments

X	A data matrix, with rows being the variables and columns being samples.
y	Groups or survival object (for cox regression).
fun	*fun* can either be a character or a function. *fun* should be one of the 't.test', 'cox', 'log2fc', and 'kruskal.test' when it is a character. findIPs() incorporates four widely used ranking criteria: t-test, univariate cox model, log2fc, and kruskal test, whose outputs are p values except log2fc (absolute log2 fold changes). The features would be ordered by specifying the argument decreasing. For instance, if fun = 't.test', the decreasing = F, such that features are order by the pvalues of t.test in a increasing manner. *fun* can also be a function to obtain ranking criteria with x and y being the only input and the ranking criteria, such as p-values being the only output.
decreasing	logical. How the rank criteria are ordered? For instance, p-value should be ordered increasingly, while fold-change should be ordered decreasingly.
topN	the number of important features included for comparison.

`method`	method to summarize rank changes. It should be one of the 'adaptive', 'weightedSpearman', and 'unweighted'. Both 'adaptive' and 'weightedSpearman' are weighted rank comparison method, but former employs the weight that are adaptive to the distribution of rank changes. 'unweighted' denotes a direct comparison of ranks without considering weights.
`nCores`	the number of CPU cores used for parallel running. If nCores = NULL, a single core is used.

Value

`kappa`	The weight function's shape is controlled by kappa, which ranges from 0 to 1. Weighted rank changes are calculated using kappa, with higher values indicating more weight on top features.
`score`	The influence of each observation on feature rankings, with larger values indicating more influence.
`origRank`	The original ranking. origRank is exactly the input. Here it is re-output for visualization purposes.
`drop1Rank`	The leave-one-out rankings.
`origRankWeighted`	The weighted original ranking
`drop1RankWeighted`	The weighted leave-one-out rankings

Examples

```
data(miller05)
X <- miller05$X
y <- miller05$y

obj <- findIPs(X, y,
               fun = 't.test',
               decreasing = FALSE,
               topN = 100,
               method = 'adaptive')

par(mfrow = c(1, 3), mar = c(4, 4, 2, 2))
plotRankScatters(obj, top = TRUE)
plotAdaptiveWeights(kappa = obj$kappa,
                    n = nrow(obj$drop1Rank),
                    type = 'line',
                    ylim = NULL)
plotIPs(obj, topn = 5, ylim = NULL)

## Interop with ExpressionSet class
library(Biobase)
data(sample.ExpressionSet)
design <- phenoData(sample.ExpressionSet)$type
IPs <- findIPs(exprs(sample.ExpressionSet), design, fun = "t.test",
               method = "adaptive")
plotIPs(IPs)

## Interop with SummarizedExperiment class
library(SummarizedExperiment)
## Make a SummarizedExperiment class
```

```
sample.SummarizedExperiment <- makeSummarizedExperimentFromExpressionSet(
  sample.ExpressionSet)

design <- colData(sample.SummarizedExperiment)$type
IPs <- findIPs(assay(sample.SummarizedExperiment), design, fun = "t.test",
               method = "adaptive")
plotIPs(IPs)
```

`getdrop1ranks` *Derive ranking lists including original and leave-one-out rankings*

Description

This function calculates the original and leave-one-out feature rankings using a predefined rank method

Usage

```
getdrop1ranks(X, y, fun, decreasing = FALSE, topN = 100, nCores = NULL)
```

Arguments

`X`	A data matrix, with rows being the variables and columns being samples.
`y`	Groups or survival object (for cox regression)
`fun`	*fun* can either be a character or a function. *fun* should be one of the 't.test', 'cox', 'log2fc', and 'kruskal.test' when it is a character. `findIPs()` incorporates four widely used ranking criteria: t-test, univariate cox model, log2fc, and kruskal test, whose outputs are p values except log2fc (absolute log2 fold changes). The features would be ordered by specifying the argument `decreasing`. For instance, if `fun = 't.test'`, the `decreasing = F`, such that features are order by the pvalues of t.test in the increasing manner. *fun* can also be a function to obtain ranking criteria with x and y being the only input and the ranking criteria, such as p-values being the only output.
`decreasing`	logical. How the rank criteria are ordered? For instance, p-value should be ordered increasingly, while fold-change should be ordered decreasingly.
`topN`	the number of important features included for comparison. The top n features in the original ranking list.
`nCores`	the number of CPU cores used for parallel running. If nCores = NULL, a single core is used.

Value

`orig`	vector:,original ranking
`drop1rank`	matrix, Leave-one-out rankings

Examples

```
data(miller05)
X <- miller05$X
y <- miller05$y
obj <- getdrop1ranks(X, y,
                     fun = 't.test',
                     decreasing = FALSE,
                     topN = 100)
rks <- sumRanks(origRank = obj$origRank,
                drop1Rank = obj$drop1Rank,
                topN = 100,
                method = 'adaptive')
plotIPs(rks, topn = 5, ylim = NULL)
```

`miller05` *miller05 data*

Description

miller05 is gene expression data with 1000 genes randomly sampled from 22283 genes and 236 samples since removing the case with missing response. The data has binary and survival response. The binary response contains 58 case with p53 mutant and 193 wild type mutant. The survival response has a total of 55 events.

Usage

```
data(miller05)
```

Format

a list

Value

miller05 data, a list containing 1000 genes and binary and survival response.

References

Miller, Lance D., et al. 'An expression signature for p53 status in human breast cancer predicts mutation status, transcriptional effects, and patient survival.' Proceedings of the National Academy of Sciences 102.38 (2005): 13550-13555.doi:10.1073pnas.0506230102

Examples

```
data(miller05)
```

`plotAdaptiveWeights` *Visualize the weight function for adaptive weights*

Description

Plot the weight function for the adaptive weights with given kappa and the list length (n).

Usage

```
plotAdaptiveWeights(kappa, n, type = c("line", "points"), ylim = NULL)
```

Arguments

`kappa`	a shape parameter of the weight function.
`n`	the length list.
`type`	draw line or points. Both line and points will be plotted if type = c('line', 'points').
`ylim`	y coordinates ranges.

Value

plot based on basic graph

Examples

```
par(mfrow = c(1, 2), mar = c(4, 4, 2, 2))
plotAdaptiveWeights(kappa = 0.01, n = 100, type = 'line', ylim = c(0, 0.025))
plotAdaptiveWeights(kappa = 0.02, n = 100, type = 'line', ylim = c(0, 0.025))
```

`plotIPs` *Visualize the influential scores*

Description

Visualize influential score using lollipop plot. The function uses the output obtained from `rank.compare` or `findIPs` function.

Usage

```
plotIPs(obj, topn = 5, ylim = NULL, ...)
```

Arguments

`obj`	the object obtained from `rank.compare` or `findIPs` function.
`topn`	the top n most influential points to be labelled in the plot.
`ylim`	y coordinates ranges
`...`	other arguments

Value

plot based on basic graph

Examples

```
data(miller05)
X <- miller05$X
y <- miller05$y
obj <- getdrop1ranks(X, y,
                     fun = 't.test',
                     decreasing = FALSE,
                     topN = 100)
rks <- sumRanks(origRank = obj$origRank,
                drop1Rank = obj$drop1Rank,
                topN = 100,
                method = 'adaptive')
plotIPs(rks, topn = 5, ylim = NULL)
```

`plotRankScatters` *Visualize the unweighted rank changes*

Description

Visualize the unweighted rank changes using scatter plot. The plot displays the original ranking and leave-one-out rankings.

Usage

```
plotRankScatters(obj, top = TRUE, points.arg = list(), top.arg = list())
```

Arguments

`obj`	the objective obtained from `findIPs()` or `sumRanks()` functions
`top`	logical, whether the most influential case needs to be plot in black
`points.arg`	a list. Arguments in `graphics::points()` can be used to define the points.
`top.arg`	a list. Arguments in `graphics::points()` can be used to define the top points.

Value

a plot based on basic graphic.

Examples

```
data(miller05)
X <- miller05$X
y <- miller05$y

obj <- getdrop1ranks(X, y,
                     fun = 't.test',
                     decreasing = FALSE,
                     topN = 100)
```

```
rks <- sumRanks(origRank = obj$origRank,
                drop1Rank = obj$drop1Rank,
                topN = 100,
                method = 'adaptive')
plotRankScatters(rks)
```

`sumRanks` *Summarize the weighted rank changes caused by case-deletion*

Description

This function measures the overall rank changes due to case deletion. A large rank changes indicates more influence of the deleted case on feature rankings. `sumRanks()` provides three methods to compute the overall rank changes: unweighted, weighted Spearman, and adaptive weights.

Usage

```
sumRanks(origRank, drop1Rank, topN = NULL, method = "adaptive", ...)
```

Arguments

`origRank`	vectors, reference rankings. For influential observation detection, origRank denotes the original ranking obtained using the whole data.
`drop1Rank`	matrix or data.frame, Each column is a feature list with a case removed.
`topN`	the top n features in origRank will be used for rank comparison. If null, include all features.
`method`	method to summarize rank changes. It should be one of the 'adaptive', 'weightedSpearman', and 'unweighted'. Both 'adaptive' and 'weightedSpearman' are weighted rank comparison method, but former employs the weight that are adaptive to the distribution of rank changes. 'unweighted' denotes a direct comparison of ranks without considering weights.
`...`	other arguments

Value

`kappa`	The weight function's shape is controlled by kappa, which ranges from 0 to 1. Weighted rank changes are calculated using kappa, with higher values indicating more weight on top features.
`score`	The influence of each observation on feature rankings, with larger values indicating more influence.
`origRank`	The original ranking. origRank is exactly the input. Here it is re-output for visualization purposes.
`drop1Rank`	The leave-one-out rankings.
`origRankWeighted`	The weighted original ranking. origRankWeighted will be returned when method = 'adaptive'.
`drop1RankWeighted`	The weighted leave-one-out rankings. drop1RankWeighted will be returned when method = 'adaptive'.

Examples

```
data(miller05)
X <- miller05$X
y <- miller05$y
obj <- getdrop1ranks(X, y,
                     fun = 't.test',
                     decreasing = FALSE,
                     topN = 100)

rks <- sumRanks(origRank = obj$origRank,
                drop1Rank = obj$drop1Rank,
                topN = 100,
                method = 'adaptive')

plotIPs(rks, topn = 5, ylim = NULL)
```

Index