Package: iSEEtree (via r-universe)

September 12, 2024

Version 0.99.6

Title Interactive visualisation for microbiome data

Description iSEEtree is an extension of iSEE for the
TreeSummarizedExperiment. It leverages the functionality from
the miaViz package for microbiome data visualisation to create
panels that are specific for TreeSummarizedExperiment objects.
Not surprisingly, it also depends on the generic panels from
iSEE.

biocViews Microbiome, Software, Visualization, GUI, ShinyApps

License Artistic-2.0

Encoding UTF-8

Depends R (>= 4.3.0), iSEE

Imports grDevices, methods, miaViz, S4Vectors, shiny, mia,
shinyWidgets, SingleCellExperiment, SummarizedExperiment,
TreeSummarizedExperiment, utils

Suggests BiocStyle, knitr, RefManageR, remotes, rmarkdown, scater,
testthat (>= 3.0.0), vegan

URL https://github.com/microbiome/iSEEtree

BugReports https://github.com/microbiome/iSEEtree/issues

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://bioc.r-universe.dev

RemoteUrl https://github.com/bioc/iSEEtree

RemoteRef HEAD

RemoteSha bd90d0dbaa5ea83075293ca9c260ceb38f2c8dbc

Contents

AbundanceDensityPlot   *Abundance density plot*

Description

Density abundance profile of single features in a `TreeSummarizedExperiment`. The panel implements `plotAbundanceDensity` to generate the plot.

Value

The AbundanceDensityPlot(...) constructor creates an instance of an AbundanceDensityPlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- `layout`, a string specifying abundance layout (jitter, density or points).
- `assay.type`, a string specifying the assay to visualize.
- `n`, a number indicating the number of top taxa to visualize.
- `flipped`, a logical specifying if the axis should be switched.
- `order_descending`, a string specifying the descending order.

In addition, this class inherits all slots from its parent Panel class.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Add relabundance assay
tse <- transformAssay(tse, method = "relabundance")
```

```
# Store panel into object
panel <- AbundanceDensityPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

AbundancePlot *Abundance plot*

Description

Composite abundance profile of all features in a [TreeSummarizedExperiment](#) object. The panel
implements [plotAbundance](#) to generate the plot.

Value

The AbundancePlot(...) constructor creates an instance of an AbundancePlot class, where any
slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- rank, a string specifying the taxonomic rank to visualize.
- use_relative, a logical indicating if the relative values should be calculated.
- add_legend, a logical indicating if the color legend should appear.

In addition, this class inherits all slots from its parent [Panel](#) class.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Store panel into object
panel <- AbundancePlot()
# View some adjustable parameters
head(slotNames(panel))
```

```
# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

iSEE   *iSEE layout for TreeSE*

Description

Panel configuration tuned to the specific properties of [TreeSummarizedExperiment](#).

Usage

```
iSEE(
  se,
  initial = NULL,
  extra = NULL,
  colormap = ExperimentColorMap(),
  landingPage = createLandingPage(),
  tour = NULL,
  appTitle = NULL,
  runLocal = TRUE,
  voice = FALSE,
  bugs = FALSE,
  saveState = NULL,
  ...
)
```

Arguments

se	A [SummarizedExperiment](#) object, ideally with named assays. If missing, an app is launched with a landing page generated by the landingPage argument.
initial	A list of [Panel](#) objects specifying the initial state of the app. The order of panels determines the sequence in which they are laid out in the interface. Defaults to one instance of each panel class available from **iSEE**.
extra	A list of additional [Panel](#) objects that might be added after the app has started. Defaults to one instance of each panel class available from **iSEE**.
colormap	An [ExperimentColorMap](#) object that defines custom colormaps to apply to individual assays, colData and rowData covariates.
landingPage	A function that renders a landing page when iSEE is started without any specified se. Ignored if se is supplied.
tour	A data.frame with the content of the interactive tour to be displayed after starting up the app. Ignored if se is not supplied.

appTitle	A string indicating the title to be displayed in the app. If not provided, the app displays the version info of [iSEE]().
runLocal	A logical indicating whether the app is to be run locally or remotely on a server, which determines how documentation will be accessed.
voice	A logical indicating whether the voice recognition should be enabled.
bugs	Set to TRUE to enable the bugs Easter egg. Alternatively, a named numeric vector control the respective number of each bug type (e.g., c(bugs=3L, spiders=1L)).
saveState	A function that accepts a single argument containing the current application state and saves it to some appropriate location.
...	Further arguments to pass to [shinyApp]().

Details

Configuring the initial state of the app is as easy as passing a list of [Panel]() objects to initial. Each element represents one panel and is typicall constructed with a command like [ReducedDimensionPlot](). Panels are filled from left to right in a row-wise manner depending on the available width. Each panel can be easily customized by modifying the parameters in each object.

The extra argument should specify [Panel]() classes that might not be shown during initialization but can be added interactively by the user after the app has started. The first instance of each new class in extra will be used as a template when the user adds a new panel of that class. Note that initial will automatically be appended to extra to form the final set of available panels, so it is not strictly necessary to re-specify instances of those initial panels in extra. (unless we want the parameters of newly created panels to be different from those at initialization).

Value

The iSEE method for the TreeSE container returns a default set of panels typically relevant for microbiome data. This configuration can be modified by defining a different set of initial panels. By default, the interface includes the following panels:

- RowDataTable()
- ColumnDataTable()
- RowTreePlot()
- AbundancePlot()
- AbundanceDensityPlot()
- ReducedDimensionPlot()
- ComplexHeatmapPlot()

Setting up a tour

The tour argument allows users to specify a custom tour to walk their audience through various panels. This is useful for describing different aspects of the dataset and highlighting interesting points in an interactive manner.

We use the format expected by the rintrojs package - see [https://github.com/carlganz/rintrojs#usage]() for more information. There should be two columns, element and intro, with the former describing the element to highlight and the latter providing some descriptive text. The [defaultTour]() also provides the default tour that is used in the Examples below.

Creating a landing page

If se is not supplied, a landing page is generated that allows users to upload their own RDS file to initialize the app. By default, the maximum request size for file uploads defaults to 5MB (https://shiny.rstudio.com/reference/shiny/0.14/shiny-options.html). To raise the limit (e.g., 50MB), run options(shiny.maxRequestSize=50*1024^2).

The landingPage argument can be used to alter the landing page, see createLandingPage for more details. This is useful for creating front-ends that can retrieve SummarizedExperiments from a database on demand for interactive visualization.

Saving application state

If users want to record the application state, they can download an RDS file containing a list with the entries:

- memory, a list of Panel objects containing the current state of the application. This can be directly re-used as the initial argument in a subsequent iSEE call.
- se, the SummarizedExperiment object of interest. This is optional and may not be present in the list, depending on the user specifications.
- colormap, the ExperimentColorMap object being used. This is optional and may not be present in the list, depending on the user specifications.

We can also provide a custom function in saveState that accepts a single argument containing this list. This is most useful when iSEE is deployed in an enterprise environment where sessions can be saved in a persistent location; combined with a suitable landingPage specification, this allows users to easily reload sessions of interest. The idea is very similar to Shiny bookmarks but is more customizable and can be used in conjunction with URL-based bookmarking.

Examples

```
# Import TreeSE
library(mia)
data("GlobalPatterns", package = "mia")
tse <- GlobalPatterns

# Agglomerate TreeSE by Genus
tse_genus <- agglomerateByRank(tse,
                               rank = "Genus",
                               onRankOnly = TRUE)

# Add relabundance assay
tse_genus <- transformAssay(tse_genus, method = "relabundance")

# Launch iSEE with custom initial panels
if (interactive()) {
  iSEE(tse_genus, initial = c(RowTreePlot(), AbundancePlot(), AbundanceDensityPlot()))
}
```

RDAPlot   *RDA plot*

Description

CCA/RDA plot for the rows of a TreeSummarizedExperiment object. The reduced dimension can be produced with runRDA and gets stored in the reducedDim slot of the experiment object. The panel implements plotRDA to generate the plot.

Value

The RDAPlot(...) constructor creates an instance of a RDAPlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- add.ellipse, a string specifying ellipse layout (filled, coloured or absent).
- colour_by, a string specifying the parameter to color by.
- add.vectors, a logical indicating if vectors should appear in the plot.
- vec.text, a logical indicating if text should be encased in a box.
- confidence.level, a numeric between 0 and 1 to adjust confidence level.
- ellipse.alpha, a numeric between 0 and 1 t o adjust ellipse opacity.
- ellipse.linewidth, a numeric specifying the size of ellipses.
- ellipse.linetype, a numeric specifying the style of ellipses.
- vec.size, a numeric specifying the size of vectors.
- vec.colour, a string specifying the colour of vectors.
- vec.linetype, a numeric specifying the style of vector lines.
- arrow.size, a numeric specifying the size of arrows.
- label.colour, a string specifying the colour of text and labels.
- label.size, a numeric specifying the size of text and labels.
- add.significance, a logical indicating if variance and p-value should appear in the labels.
- add.expl.var, a logical indicating if variance should appear on the coordinate axes.

In addition, this class inherits all slots from its parent Panel class.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("enterotype", package = "mia")
tse <- enterotype

# Run RDA and store results into TreeSE
tse <- runRDA(tse,
              formula = assay ~ ClinicalStatus + Gender + Age,
              FUN = vegan::vegdist,
              distance = "bray",
              na.action = na.exclude)

# Store panel into object
panel <- RDAPlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

RowTreePlot   *Row tree plot*

Description

Hierarchical tree for the rows of a TreeSummarizedExperiment object. The tree can be produced with addTaxonomyTree and gets stored in the rowTree slot of the experiment object. The panel implements plotRowTree to generate the plot.

Value

The RowTreePlot(...) constructor creates an instance of a RowTreePlot class, where any slot and its value can be passed to ... as a named argument.

Slot overview

The following slots control the thresholds used in the visualization:

- layout, a string specifying tree layout

- add_legend, a logical indicating if color legend should appear.

- edge_colour_by, a string specifying parameter to color lines by when colour_parameters = "Edge".

- edge_size_by, a string specifying parameter to size lines by when size_parameters = "Edge".

- tip_colour_by, a string specifying parameter to color tips by when colour_parameters = ″Tip″.

- tip_size_by, a string specifying parameter to size tips by when size_parameters = ″Tip″.

- tip_shape_by, a string specifying parameter to shape tips by when shape_parameters = ″Tip″.

- node_colour_by, a string specifying parameter to color nodes by when colour_parameters = ″Node″.

- node_size_by, a string specifying parameter to size nodes by when size_parameters = ″Node″.

- node_shape_by, a string specifying parameter to shape nodes by when shape_parameters = ″Node″.

- order_tree, a logical indicating if tree is ordered by alphabetic order of taxonomic levels.

In addition, this class inherits all slots from its parent Panel class.

Author(s)

Giulio Benedetti

Examples

```
# Import TreeSE
library(mia)
data("Tengeler2020", package = "mia")
tse <- Tengeler2020

# Store panel into object
panel <- RowTreePlot()
# View some adjustable parameters
head(slotNames(panel))

# Launch iSEE with custom initial panel
if (interactive()) {
  iSEE(tse, initial = c(panel))
}
```

Index