Package 'NAIR'

March 24, 2023

Type Package

Title Network Analysis for Immune Repertoire

Version 0.0.9035

Maintainer Brian Neal <Brian.Neal@ucsf.edu>

Description Pipelines for studying the immune repertoire of T and B cells via network analysis based on the similarity of their receptor sequences. Relate clinical outcomes to immune repertoires based on their network properties, or to particular clusters and clones within a repertoire.

License GPL (>= 3)

Encoding UTF-8

LazyData true

Imports Rcpp (>= 1.0.8), Matrix, reticulate, igraph, ggraph, ggplot2, stats, utils, grDevices, plyr, dplyr, reshape2, gplots, RColorBrewer, viridisLite, graphics

LinkingTo Rcpp (>= 1.0.8), RcppArmadillo (>= 0.10.8.0.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/reticulate list(
packages = list(
list(package = ``numpy''),
list(package = ``pandas''),
list(package = ``tensorflow''),
list(package = ``keras'')
)
)

R topics documented:

`addClusterLabels` *Label Notable Clusters in a Graph Plot*

Description

Given a graph plot and clustering analysis output, ranks clusters according to specified property (node size by default) and labels those above a certain rank with their cluster IDs, returning the labeled plot.

Usage

```
addClusterLabels(
  plot, net,
  top_n_clusters = 20,
  criterion = "node_count",
  size = 5, color = "black")
```

Arguments

`plot`	A `ggraph` object.
`net`	A list of network objects containing list items `node_data` and `cluster_data`, such as the output of `buildRepSeq` when called with `cluster_stats = TRUE`.
`top_n_clusters`	A positive integer specifying the number of clusters to label. Those with the highest rank according to the `criterion` argument will be labeled.

criterion	The name of a numeric cluster property by which to rank the clusters. See getClusterStats for a list of properties. The default is "node_size".
size	A positive value specifying the size of the text labels.
color	The color of the text labels.

Value

A ggraph object; the input plot, annotated with cluster ID labels according to the user specifications.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network, but don't print the graph plot
network <- buildRepSeqNetwork(
  toy_data, seq_col = "CloneSeq",
  node_stats = TRUE, cluster_stats = TRUE,
  color_nodes_by = "transitivity", color_scheme = "plasma-1",
  size_nodes_by = "degree", node_size_limits = c(0.5, 1.5),
  print_plots = FALSE, output_dir = NULL)

# Add labels to the two largest clusters and print the plot
addClusterLabels(plot = network$plots$transitivity,
                 net = network,
                 top_n_clusters = 2,
                 criterion = "node_count" # (the default)
                 )

## End(Not run)
```

---

addClusterMembership *Compute Cluster Membership of Network Nodes*

Description

Given immune repertoire sequencing data and the igraph object for the network by sequence similarity, identifies dense clusters within the network and augments the data with a cluster membership variable.

Usage

```
addClusterMembership(data, net, fun = cluster_fast_greedy)
```

Arguments

data	A data frame containing the immune repertoire sequence data, with clones/cells indexed by row.
net	An igraph object containing the network on the rows of the data.
fun	A function from the igraph package that takes a graph as its primary argument, implements a community-detection algorithm and returns a community object. See details below.

Details

The clusters are identified by searching for densely-connected subgraphs (with relatively many edge connections among their nodes), also called communities, within the network.

The fun argument controls the algorithm used to detect the community structure in the network. Following is a list of all functions in the igraph package (as of version 1.3.5) that can be used: cluster_fast_greedy, cluster_edge_betweenness, cluster_fluid_communities, cluster_infomap, cluster_label_prop, cluster_leading_eigen, cluster_leiden, cluster_louvain, cluster_optimal, cluster_spinglass, cluster_walktrap.

Value

A copy of data with an additional column named cluster_id that contains the cluster membership for each row.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Add cluster ID
data_w_clusterID <- addClusterMembership(net$node_data, net$igraph)
```

---

addGraphLabels *Label Nodes on a Graph Plot*

Description

Annotate a graph plot to add custom labels to the nodes.

Usage

```
addGraphLabels(plot, node_labels, size = 5, color = "black")
```

Arguments

plot	A ggraph object.
node_labels	A vector of node labels (the length should match the number of nodes in the plot).
size	The size of the node labels.
color	The color of the node labels.

Details

Labels are added via ggraph::geom_node_text.

Value

A ggraph object; the input plot with labels added.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData(samples = 1, sample_size = 10,
                            prefix_length = 1)

# Generate network, but don't print the plot yet
network <- buildRepSeqNetwork(
  toy_data, seq_col = "CloneSeq",
  print_plots = FALSE, output_dir = NULL)

# Annotate the plot, labeling each node with its clone sequence
network$plots[[1]] <- addGraphLabels(network$plots[[1]],
                                     network$node_data$CloneSeq,
                                     size = 4)

# View the plot
network$plots[[1]]
```

---

addNodeNetworkStats *Compute Node-Level Network Statistics*

Description

Given immune repertoire sequence data and the corresponding repertoire network, compute the specified node-level network statistics, which are appended as additional columns to the rep-seq data before it is returned.

Usage

```
addNodeNetworkStats(data, net,
                    stats_to_include = chooseNodeStats(),
                    cluster_fun = cluster_fast_greedy)
```

Arguments

data	A data frame containing the immune repertoire sequence data, with clones indexed by row.
net	An igraph object containing the network corresponding to the clones in data.
stats_to_include	A list generated using chooseNodeStats that specifies the network statistics to compute. Alternatively, stats_to_include = "all" will compute all node-level network stats, and stats_to_include = "cluster_id_only" will compute cluster membership only.
cluster_fun	Passed to addClusterMembership. Controls the algorithm used for cluster identification.

Value

A data frame with the same rows and columns as data, with additional columns containing the network statistics for the clone in each row.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Add default network statistics
data <- addNodeNetworkStats(net$node_data, net$igraph)

## End(Not run)
```

`adjacencyMatAtchleyFromSeqs` *Network Adjacency Matrix Based on Clone Sequence Atchley Factor Representations*

Description

Given a list of T-cell receptor CDR3 amino acid sequences, encodes each sequence as a 30-dimensional numeric vector based on the Atchley factor representation of its terms using a trained encoder, then computes the corresponding network adjacency matrix based on the Euclidean distance between encoded vectors using a specified cutoff.

Usage

```
adjacencyMatAtchleyFromSeqs(
    seqs,
    contig_ids = seq_along(seqs),
    max_dist,
    return_type = "adjacency_matrix",
    outfile_distance_matrix = NULL)
```

Arguments

`seqs`	A character vector containing the TCR CDR3 amino acid sequences.
`contig_ids`	A numeric vector of the same length as `seqs`, used to uniquely identify its elements. By default, this is simply the sequence of integers from 1 to `length(seqs)`.
`max_dist`	A positive integer specifying the maximum Euclidean distance at which the numeric vectors corresponding to two TCR sequences are concidered adjacent (joined by an edge) in the network graph.
`return_type`	A character string specifying the type of matrix to be returned. Valid options are `"adjacency_matrix"` and `"distance_matrix"`, the latter of which returns the matrix containing the pairwise Euclidean distances rather than the pairwise adjacencies.
`outfile_distance_matrix`	An optional argument specifying a `csv` file to which the matrix will be written.

Details

The adjacency matrix of a graph with n nodes/vertices is the symmetric $n \times n$ matrix for which the (i, j)th entry is equal to 1 if nodes i and j are adjacent (joined by an edge in the graph) and 0 otherwise.

Each element of `seqs` represents a node in the corresponding network graph; two nodes are adjacent if the Euclidean distance between their numerically encoded values is at most `max_dist`.

Value

A matrix.

Note

The encoder was trained specifically on TCR CDR3 amino acid sequences and is not appropriate for use with other amino acid sequences.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
cdr3 <- c("CASSEAQGSGSTDTQYF",
          "CATTEGSNTGELFF",
          "CASSIGDNEQFF",
          "CATSRDPDRGQSDTQYF",
          "CASSPTGLSGNTIYF",
          "CASSEEAGKDTQYF",
          "CASSGGADTQYF",
          "CASSLGLATDTQYF",
          "CASSEKEEVGELFF",
          "CASSSRTSGGAGELFF")
adjacencyMatAtchleyFromSeqs(
    cdr3, max_dist = 3)

adjacencyMatAtchleyFromSeqs(
    cdr3,
    return_type = "distance_matrix")

## End(Not run)
```

---

aggregateIdenticalClones

Aggregate Counts/Frequencies for Clones With Identical Receptor Sequences

---

Description

Given bulk TCR/BCR repertoire sequence data with clones indexed by row, aggregates rows with identical receptor sequences. Clone count and frequency are summed, other information is discarded and a column is added to record the number of unique clones (rows) for each unique receptor sequence.

Usage

```
aggregateIdenticalClones(data, clone_col, count_col, freq_col,
                         grouping_cols = NULL)
```

Arguments

data	A data frame containing the immune repertoire sequence data, with clones indexed by row.
clone_col	The column name or number of data containing the receptor sequences (e.g., TCR CDR3 nucleotide sequence or amino acid sequence).
count_col	The column name or number of data containing the clone counts.
freq_col	The column name or number of data containing the clone frequencies.
grouping_cols	An optional charcter or integer vector specifying one or more columns of data, each of which is treated as a group/label variable. If supplied, clones belonging to distinct groups will be treated as having distinct receptor sequences, effectively aggregating only the identical clones within each group.

Value

A data frame whose first column contains the receptor sequences and has the same name as the column of data specified by clone_col. If additional columns of data were supplied via the grouping_cols argument, these columns will also be inherited. The remaining columns are as follows:

AggregatedCloneCount
: The aggregated clone count for each receptor sequence.

AggregatedCloneFrequency
: The aggregated clone frequency for each receptor sequence.

UniqueCloneCount
: The number of clones (rows) in data possessing the receptor sequence for the current row (if groups are supplied, this is a within-group count).

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Create some data
data <- data.frame(
  clone_seq = c("ATCG", rep("ACAC", 2), rep("GGGG", 4)),
  clone_count = rep(1, 7),
  clone_freq = rep(1 / 7, 7),
  # group/label variable 1
  time_point = c("t_0", rep(c("t_0", "t_1"), 3)),
  # group/label variable 2
  subject_id = c(rep(1, 5), rep(2, 2))
)

# Aggregate clones by receptor sequence (default usage)
data_agg <- aggregateIdenticalClones(
  data, "clone_seq", "clone_count", "clone_freq")

# Aggregate clones by receptor sequence and time point
```

```
data_agg_time <- aggregateIdenticalClones(
  data, "clone_seq", "clone_count", "clone_freq",
  grouping_cols = "time_point")

# Aggregate clones by receptor sequence and subject
data_agg_subject <- aggregateIdenticalClones(
  data, "clone_seq", "clone_count", "clone_freq",
  grouping_cols = "subject_id")

# Aggregate clones by receptor sequence, subject & time point
# (note all clones in each group are already unique)
data_agg_time_subject <- aggregateIdenticalClones(
  data, "clone_seq", "clone_count", "clone_freq",
  grouping_cols = c("subject_id", "time_point"))
```

`buildAssociatedClusterNetwork` *Combine Associated Sequence Neighborhoods and Build Network*

Description

After running `findAssociatedClones`, this function is used to combine some or all of the resulting neighborhoods into a single network in which clustering and network analysis are performed.

Usage

```
buildAssociatedClusterNetwork(
    file_list, input_type = "csv",
    data_symbols = NULL,
    header = TRUE, sep = ",",
    seq_col,
    min_seq_length = NULL, drop_matches = NULL,
    drop_isolated_nodes = FALSE,
    node_stats = TRUE,
    stats_to_include =
      chooseNodeStats(cluster_id = TRUE),
    cluster_stats = TRUE,
    color_nodes_by = "cluster_id",
    output_name = "AssociatedClusterNetwork",
    ...
)
```

Arguments

`file_list`	Passed to `loadDataFromFileList`.
`input_type`	Passed to `loadDataFromFileList`.
`data_symbols`	Passed to `loadDataFromFileList`.
`header`	Passed to `loadDataFromFileList`.
`sep`	Passed to `loadDataFromFileList`.
`seq_col`	Passed to `buildRepSeqNetwork`.

min_seq_length	Passed to buildRepSeqNetwork.
drop_matches	Passed to buildRepSeqNetwork.
drop_isolated_nodes	Passed to buildRepSeqNetwork.
node_stats	Passed to buildRepSeqNetwork.
stats_to_include	Passed to buildRepSeqNetwork.
cluster_stats	Passed to buildRepSeqNetwork.
color_nodes_by	Passed to buildRepSeqNetwork.
output_name	Passed to buildRepSeqNetwork.
...	Other arguments to buildRepSeqNetwork.

Details

Essentially a wrapper to loadDataFromFileList and buildRepSeqNetwork, with default argument values tailored to the associated cluster workflow.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

findAssociatedSeqs findAssociatedClones

Examples

```
## Not run:
## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)

samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
```

```
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## 1. Find Associated Sequences

# input files for step 1 (one per sample)
input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)

# group labels for the samples
group_labels <- c(rep("reference", samples / 2), rep("comparison", samples / 2))

# search across samples for associated sequences using Fisher's exact test
associated_seqs <- findAssociatedSeqs(
  file_list = input_files, input_type = "rds",
  group_ids = group_labels, groups = c("reference", "comparison"),
  min_seq_length = NULL, drop_matches = NULL,
  seq_col = "CloneSeq", outfile = NULL)
head(associated_seqs)


## 2. Find Associated Clones

# output directory for current step
dir_nbds <- file.path(data_dir, "assoc_seq_nbds")

# Identify neighborhood around each associated sequence
findAssociatedClones(
  file_list = input_files, input_type = "rds", group_ids = group_labels,
  seq_col = "CloneSeq", dist_type = "levenshtein",
  assoc_seqs = associated_seqs$ReceptorSeq,
  min_seq_length = NULL, drop_matches = NULL,
  output_dir = dir_nbds)


## 3. Build Associated Cluster Network

# Files created during previous step
nbd_files <- list.files(dir_nbds, full.names = TRUE)
nbd_files
```

```
# Combine neighborhoods and perform network analysis
all_clusters <- buildAssociatedClusterNetwork(
  file_list = nbd_files,
  seq_col = "CloneSeq", dist_type = "levenshtein", size_nodes_by = 1.5,
  output_dir = file.path(data_dir, "assoc_clusters"))

# focus on a particular cluster
buildRepSeqNetwork(
  data = all_clusters$node_data[all_clusters$node_data$cluster_id == 3, ],
  seq_col = "CloneSeq", color_nodes_by = "CloneSeq", size_nodes_by = 2,
  output_dir = NULL, output_name = "Cluster 3")

## End(Not run)
```

---

buildPublicClusterNetwork

Build Network of Public Clusters

Description

Given node-level meta data for each sample's filtered public clusters, combine the data across samples and perform network analysis.

Usage

```
buildPublicClusterNetwork(

  ## Input ##
  file_list =
    list.files(file.path(getwd(), "public_clusters", "node_meta_data")),
  input_type = "rds", data_symbols = "ndat", header = TRUE, sep = "",
  seq_col,

  ## Network ##
  drop_isolated_nodes = FALSE,
  node_stats = TRUE, stats_to_include = "all", cluster_stats = TRUE,

  ## Visualization ##
  color_nodes_by = c("ClusterIDPublic", "SampleID"),
  color_scheme = "turbo", color_title = c("public cluster", "sample"),

  ## Output ##
  output_dir = file.path(getwd(), "public_clusters"),
  output_name = "PublicClusterNetwork",

  ...

)
```

Arguments

file_list	Passed to loadDataFromFileList when loading the node-level meta data for each sample (one data frame per sample).
input_type	Passed to loadDataFromFileList.
data_symbols	Passed to loadDataFromFileList.
header	Passed to loadDataFromFileList.
sep	Passed to loadDataFromFileList.
seq_col	The column name or number of each sample's data frame that contains the receptor sequences to be used as the basis of similarity between rows during network analysis. This column must have the same name in each sample.
drop_isolated_nodes	Passed to buildRepSeqNetwork.
node_stats	Passed to buildRepSeqNetwork.
stats_to_include	Passed to buildRepSeqNetwork.
cluster_stats	Passed to buildRepSeqNetwork.
color_nodes_by	Passed to buildRepSeqNetwork.
color_scheme	Passed to buildRepSeqNetwork.
color_title	Passed to buildRepSeqNetwork.
output_dir	Passed to buildRepSeqNetwork.
output_name	Passed to buildRepSeqNetwork.
...	Other arguments to buildRepSeqNetwork.

Details

The input data is intended to be obtained using findPublicClusters() on the unfiltered RepSeq data for each sample.

Any node-level properties computed are renamed to reflect their association to the public network. The names used are ClusterIDPublic, PublicNetworkDegree, PublicTransitivity, PublicCloseness, PublicCentralityByCloseness, PublicEigenCentrality, PublicCentralityByEigen, PublicBetweenness, PublicCentralityByBetweenness, PublicAuthorityScore, PublicCoreness, PublicPageRank.

Value

A list of network objects as returned by buildRepSeqNetwork().

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

findPublicClusters buildPublicClusterNetworkByRepresentative

Examples

```
## Not run:

## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)

samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## 1. Find Public Clusters in Each Sample

input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)
dir_filtered_samples <- file.path(data_dir, "filtered_samples")
findPublicClusters(
  file_list = input_files, input_type = "rds",
  sample_ids = paste0("Sample", 1:samples),
  seq_col = "CloneSeq", count_col = "CloneCount",
  min_seq_length = NULL, drop_matches = NULL,
  output_dir = dir_filtered_samples)
```

```
## 2. Build Public Cluster Network

# Node-level meta data for each sample's public clusters
dir_filtered_samples_node <- file.path(dir_filtered_samples, "node_meta_data")
files_filtered_samples_node <- list.files(dir_filtered_samples_node,
                                          full.names = TRUE)
head(files_filtered_samples_node)

dir_out <- file.path(data_dir, "public_clusters")

# Collect clones from all public clusters and perform network analysis
buildPublicClusterNetwork(
  file_list = files_filtered_samples_node,
  seq_col = "CloneSeq", count_col = "CloneCount",
  color_nodes_by = "cluster_id", output_dir = dir_out)

## End(Not run)
```

---

buildPublicClusterNetworkByRepresentative
Build Public Cluster Network By Representative

Description

Given cluster-level meta data for each sample's filtered public clusters, combine the data across samples and perform network analysis using a representative sequence from each cluster.

Usage

```
buildPublicClusterNetworkByRepresentative(

  ## Input ##
  file_list,
  input_type = "rds", data_symbols = "cdat",
  header = TRUE, sep = "",
  seq_col = "seq_w_max_count",
  count_col = "agg_count",

  ## Network ##
  dist_type = "hamming",
  dist_cutoff = 1,
  cluster_fun = cluster_fast_greedy,

  ## Visualization ##
  plots = TRUE,
  print_plots = TRUE,
  plot_title = "auto",
  plot_subtitle = "auto",
  color_nodes_by = c("ClusterIDPublic", "SampleID"),
  color_scheme = "turbo",
```

```
  color_title = c("public cluster", "sample"),
  ...,

  ## Output ##
  output_dir = file.path(getwd(), "public_clusters"),
  output_type = "rda",
  output_name = "PubClustByRepresentative",
  pdf_width = 12, pdf_height = 10

)
```

Arguments

file_list	Passed to loadDataFromFileList when loading the cluster-level meta data for each sample (one data frame per sample). Data frames should match the format of the cluster-level meta data returned by buildRepSeqNetwork.
input_type	Passed to loadDataFromFileList.
data_symbols	Passed to loadDataFromFileList.
header	Passed to loadDataFromFileList.
sep	Passed to loadDataFromFileList.
seq_col	The column name or number of each data frame that contains the representative sequence to be used as the basis of similarity between rows during network analysis. This column must have the same name in each sample.
count_col	Passed to buildRepSeqNetwork.
dist_type	Passed to buildRepSeqNetwork.
dist_cutoff	Passed to buildRepSeqNetwork.
cluster_fun	Passed to addClusterMembership. Controls the algorithm used for cluster identification.
plots	Passed to buildRepSeqNetwork.
print_plots	Passed to buildRepSeqNetwork.
plot_title	Passed to buildRepSeqNetwork.
plot_subtitle	Passed to buildRepSeqNetwork.
color_nodes_by	Passed to generateNetworkGraphPlots.
color_scheme	Passed to generateNetworkGraphPlots.
color_title	Passed to generateNetworkGraphPlots.
...	Other arguments to generateNetworkGraphPlots.
output_dir	Passed to saveNetwork.
output_type	Passed to saveNetwork.
output_name	Passed to saveNetwork.
pdf_width	Passed to saveNetwork.
pdf_height	Passed to saveNetwork.

Details

This function is intended for performing network analysis on the public clusters obtained using `findPublicClusters()`.

By using the cluster-level meta data as the input to `buildRepSeqNetwork()`, this function treats each public cluster as a single node and performs network analysis with similarity based on a representative sequence for each cluster (e.g., the sequence with the greatest clone count). `buildRepSeqNetwork` is called without any filtering and with `drop_isolated_nodes = FALSE`.

All node-level properties are automatically computed for the network, and cluster-level properties for the network (in which clusters of nodes represent clusters of public-cluster-representatives) are computed based on the properties of the public cluster represented by each node. See the 'value' section for a list and description of each property.

Value

A list of network objects as per `buildRepSeqNetwork()`. The data frame `cluster_data` includes the following variables:

`TotalSampleLevelNodes`
: Each network node in the cluster for the current row represents a public cluster; this value is the sum of the sample-network-level nodes contained in all such public clusters for the current row.

`TotalCloneCount`
: Each network node in the cluster for the current row represents a public cluster; this value is the sum of the aggregate clone count across all such public clusters for the current row.

`MeanOfMeanSeqLength`
: Each network node in the cluster for the current row represents a public cluster; this value is the mean value of the mean sequence length across all such public clusters for the current row.

`MeanDegreeInPublicNet`
: The mean network degree of all the network nodes in the cluster for the current row.

`MaxDegreeInPublicNet`
: The maximum network degree of all the network nodes in the cluster for the current row.

`SeqWithMaxDegree`
: The representative sequence of the network node with maximum network degree of all the network nodes in the cluster for the current row. If more than one node attains the maximum network degree, the first sequence found is returned.

`MaxCloneCount`
: Each network node in the cluster for the current row represents a public cluster; this is the maximum value of the maximum clone count property across all such public clusters for the current row.

`SampleWithMaxCloneCount`
: The sample possessing the public cluster with the maximum value of the maximum clone count property across all the public clusters represented by the network nodes in the cluster for the current row. If more than one public cluster attains the maximum value, the first sample ID found is returned.

`SeqWithMaxCloneCount`
: The representative sequence of the public cluster with the maximum value of the maximum clone count property across all the public clusters represented by the network nodes in the cluster for the current row. If more than one public cluster attains the maximum value, the first sequence found is returned.

`MaxAggCloneCount`
: Each network node in the cluster for the current row represents a public cluster; this is the maximum value of the aggregate clone count property across all such public clusters for the current row.

`SampleWithMaxAggCloneCount`
: The sample possessing the public cluster with the maximum value of the aggregate clone count property across all the public clusters represented by the network nodes in the cluster for the current row. If more than one public cluster attains the maximum value, the first sample ID found is returned.

`SeqWithMaxAggCloneCount`
: The representative sequence of the public cluster with the maximum value of the aggregate clone count property across all the public clusters represented by the network nodes in the cluster for the current row. If more than one public cluster attains the maximum value, the first sequence found is returned.

`DiameterLength` See `?getClusterStats()`.

`Assortativity` See `?getClusterStats()`.

`GlobalTransitivity`
: See `?getClusterStats()`.

`EdgeDensity` See `?getClusterStats()`.

`DegreeCentralityIndex`
: See `?getClusterStats()`.

`ClosenessCentralityIndex`
: See `?getClusterStats()`.

`EigenCentralityIndex`
: See `?getClusterStats()`.

`EigenCentralityEigenvalue`
: See `?getClusterStats()`.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

`findPublicClusters` `buildPublicClusterNetwork`

Examples

```
## Not run:
## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)
```

```
samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## 1. Find Public Clusters in Each Sample

# input files for step 1 (one per sample)
input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)

# Search across samples for public clusters
dir_filtered_samples <- file.path(data_dir, "filtered_samples")
findPublicClusters(
  file_list = input_files, input_type = "rds",
  sample_ids = paste0("Sample", 1:samples),
  seq_col = "CloneSeq", count_col = "CloneCount",
  min_seq_length = NULL, drop_matches = NULL,
  output_dir = dir_filtered_samples)


## 2. Build Public Cluster Network by Representative Sequence

# Cluster-level meta data for each sample's public clusters
dir_filtered_samples_cluster <- file.path(dir_filtered_samples, "cluster_meta_data")
files_filtered_samples_cluster <- list.files(dir_filtered_samples_cluster,
                                             full.names = TRUE)
head(files_filtered_samples_cluster)

dir_out <- file.path(data_dir, "public_clusters")
```

```
buildPublicClusterNetworkByRepresentative(
  file_list = files_filtered_samples_cluster,
  color_nodes_by = "ClusterIDPublic", output_dir = dir_out,
  size_nodes_by = 1)

## End(Not run)
```

---

buildRepSeqNetwork *Immune Repertoire Network By Sequence Similarity*

Description

Builds the network graph for an immune repertoire based on sequence similarity, computes specified network properties and generates customized visualizations.

Usage

```
buildRepSeqNetwork(

  ## Input ##
  data, seq_col, count_col = NULL,
  subset_cols = NULL,
  min_seq_length = 3, drop_matches = NULL,

  ## Network ##
  dist_type = "hamming", dist_cutoff = 1,
  drop_isolated_nodes = TRUE,
  node_stats = FALSE,
  stats_to_include = chooseNodeStats(),
  cluster_stats = FALSE,
  cluster_fun = cluster_fast_greedy,

  ## Visualization ##
  plots = TRUE,
  print_plots = TRUE,
  plot_title = "auto",
  plot_subtitle = "auto",
  color_nodes_by = "auto",
  ...,

  ## Output ##
  output_dir = getwd(),
  output_type = "individual",
  output_name = "MyRepSeqNetwork",
  pdf_width = 12, pdf_height = 10

)
```

Arguments

data	A data frame containing the immune repertoire sequencing data, with variables indexed by column and observations (e.g., clones or cells) indexed by row.
seq_col	The column name or number of data containing the receptor sequences to be used as the basis of similarity between rows. Also accepts a vector of length 2 specifying distinct sequence columns (e.g., alpha, beta chains); then two rows are similar only if both types of sequences are similar.
count_col	Optional column name or number of data containing a measure of abundance, e.g., clone count. Passed to getClusterStats; only relevant if cluster_stats = TRUE.
subset_cols	Optional vector of column names or numbers of data; if supplied, only these plus other relevant columns are included in the output. If NULL (the default), all columns of data are included.
min_seq_length	A numeric scalar, or NULL. Observations whose receptor sequences have fewer than min_seq_length characters are removed prior to network analysis.
drop_matches	An optional regular expression or character string. Observations whose receptor sequences return a match are removed prior to network analysis.
dist_type	The type of function to use as a measure of similarity between two receptor sequences. Valid options are "hamming" (the default), "levenshtein" and "euclidean_on_atchley" (only applicable to TCR CDR3 amino acid sequences).
dist_cutoff	A nonnegative scalar specifying the maximum distance threshold for similarity between receptor sequences.
drop_isolated_nodes	A logical scalar; should observations whose receptor sequences are not similar to any other sequences be dropped from the network?
node_stats	A logical scalar; should node-level network properties be computed?
stats_to_include	A logical vector returned by chooseNodeStats(), specifying the node-level properties to include. Also accepts the values "all" and "cluster_id_only". Only relevant if node_stats = TRUE.
cluster_stats	A logical scalar; should cluster-level network properties be computed?
cluster_fun	Passed to addClusterMembership. Controls the algorithm used for cluster identification.
plots	A logical scalar; should plot(s) of the network graph be generated?
print_plots	A logical scalar; should visualizations be printed in the R plotting window?
plot_title	A character string (or NULL value) to be used as the title in visualizations. The default value "auto" generates the title dynamically.
plot_subtitle	A character string (or NULL value) to be used as the subtitle in visualizations. The default value "auto" generates the title dynamically.
color_nodes_by	Optional. Column name or number of data used to color network nodes in visualizations. If a vector of values is supplied, one plot will be generated for each value. The value "auto" attempts to use one of several variables (e.g., cluster ID) to color the nodes, depending on what is available.
...	Other arguments to generateNetworkGraphPlots.
output_dir	The output directory; if NULL, output will not be written to file.

output_type	A character string specifying the file format to use when writing output to file. Default "individual" saves each item as a separate, uncompressed file, with data frames saved in csv format. "rda" and "rds" save the output list as a rda and rds file, respectively. For all output types, any plots generated will also be saved in a pdf file.
output_name	A character string to be used as a common filename prefix for any files saved.
pdf_width	Passed to the width argument of grDevices::pdf().
pdf_height	Passed to the height argument of grDevices::pdf().

Details

To build the immune repertoire network, each TCR/BCR clone (bulk data) or cell (single-cell data) is modeled as a network node (corresponding to a single row of the input data). Two nodes are considered adjacent (share an edge) if their receptor sequences are sufficiently similar.

Both nucleotide and amino acid sequences are supported. Sequence similarity is based on either the Hamming distance or Levenshtein (edit) distance. For TCR CDR3 amino acid sequences, an alternative measure of similarity is also available via the argument dist_type = "euclidean_on_atchley"; this represents sequences as 30-dimensional numeric vectors according to the Atchley factors of their amino acids (which encode biological properties) using a trained encoder. Similarity is then based on the Euclidean distance between these representations. Selecting a distance threshold is challenging in this setting, however, as particular values lack the clear interpretations enjoyed by the Hamming and Levenshtein distances.

The graph adjacency matrix for the network is computed and used to generate a igraph network object. Network properties are computed largely through the use of igraph functions. The visualizations are generated with ggraph.

Value

If the constructed network contains no edges, the function will return NULL with a warning. Otherwise, the function invisibly returns a list containing some or all of the following items:

igraph	An igraph object containing the edge list for the network.
adjacency_matrix	The network graph adjacency matrix, stored as a sparse matrix of class dgCMatrix from the Matrix package.
node_data	A data frame containing the node-level meta-data for the network. This data frame contains all column names of data unless subset_cols is non-null, in which case only the column names of data specified by subset_cols, along with other relevant ones (e.g., seq_col), will be present in the returned data frame. The data frame will additionally contain columns for any node-level network properties computed (as specified through the node_stats and stats_to_include arguments). The data frame will contain one row for each node that remains in the network after all filtering has occurred, including any filtering based on drop_isolated_nodes. The row names of the original input data will be preserved.
cluster_data	A data frame containing the cluster-level properties, with one row per cluster. Only included if cluster_stats = TRUE.
plots	A list containing the graph plot(s) as ggraph objects. Only included if plots = TRUE.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:

# Simulate some toy data for demonstration
toy_data <- simulateToyData()

network <- buildRepSeqNetwork(
  toy_data, seq_col = "CloneSeq",
  node_stats = TRUE, cluster_stats = TRUE,
  color_nodes_by = "transitivity", color_scheme = "plasma-1",
  size_nodes_by = "degree", node_size_limits = c(0.5, 1.5),
  output_dir = NULL)

## End(Not run)
```

---

chooseNodeStats *Specify Node Network Statistics to Compute*

Description

Create a named logical vector specifying node-level network stats to compute. The output is passed to the `stats_to_include` argument of functions such as `buildRepSeqNetwork` and `addNodeNetworkStats`.

Usage

```
chooseNodeStats(degree = TRUE,
                cluster_id = FALSE,
                transitivity = TRUE,
                closeness = FALSE,
                centrality_by_closeness = FALSE,
                eigen_centrality = TRUE,
                centrality_by_eigen = TRUE,
                betweenness = TRUE,
                centrality_by_betweenness = TRUE,
                authority_score = TRUE,
                coreness = TRUE,
                page_rank = TRUE,
                all_stats = FALSE)

# Original name (still exists for backward compatibility)
node_stat_settings(degree = TRUE,
                   cluster_id = FALSE,
                   transitivity = TRUE,
```

```
                    closeness = FALSE,
                    centrality_by_closeness = FALSE,
                    eigen_centrality = TRUE,
                    centrality_by_eigen = TRUE,
                    betweenness = TRUE,
                    centrality_by_betweenness = TRUE,
                    authority_score = TRUE,
                    coreness = TRUE,
                    page_rank = TRUE,
                    all_stats = FALSE)
```

Arguments

degree	Logical. Whether to compute network degree.
cluster_id	Logical. Whether to perform clustering, which is done using igraph::cluster_fast_greedy(), in order to determine the cluster membership of each node.
transitivity	Logical. Whether to compute node-level network transitivity, which is done using igraph::transitivity() with type = "local" The local transitivity of a node is the the number of triangles connected to the node relative to the number of triples centered on that node.
closeness	Logical. Whether to compute network closeness, which is done using igraph::closeness().
centrality_by_closeness	Logical. Whether to compute network centrality by closeness, which is done using igraph::centr_clo()$res.
eigen_centrality	Logical. Whether to compute the eigenvector centrality scores of node network positions, which is done using igraph::eigen_centrality()$vector with weights = NA. The centrality scores correspond to the values of the first eigenvector of the adjacency matrix for the cluster graph.
centrality_by_eigen	Logical. Whether to compute node-level network centrality scores based on eigenvector centrality scores, which is done using igraph::centr_eigen()$vector.
betweenness	Logical. Whether to compute network betweenness, which is done using igraph::betweenness().
centrality_by_betweenness	Logical. Whether to compute network centrality by betweenness, which is done using igraph::centr_betw()$res.
authority_score	Logical. Whether to compute the authority score, which is done using igraph::authority_score()
coreness	Logical. Whether to compute network coreness, which is done using igraph::coreness().
page_rank	Logical. Whether to compute page rank, which is done using igraph::page_rank()$vector.
all_stats	Logical. If TRUE, then addNodeNetworkStats will disregard the values for the other settings and instead treat all of them as TRUE.

Value

A logical vector for use with addNodeNetworkStats or buildRepSeqNetwork as input to the stats_to_include argument.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Add custom network statistics
data_w_stats <-
  addNodeNetworkStats(
    net$node_data, net$igraph,
    stats_to_include =
      chooseNodeStats(
          cluster_id = TRUE,
          closeness = TRUE,
          centrality_by_closeness = TRUE,
          centrality_by_betweenness = FALSE
      )
  )

# Add all network statistics
data_w_all_stats <-
  addNodeNetworkStats(
    net$node_data, net$igraph,
    stats_to_include =
      chooseNodeStats(all_stats = TRUE)
  )

## End(Not run)
```

`combineSamples` *Load Multiple Data Files and Combine*

Description

Given multiple data frames stored in separate files, `loadDataFromFileList` loads and combines them into a single data frame.

`combineSamples` is similar to `loadDataFromFileList`, but allows the data frames to be filtered and subsetted before being combined, and can automatically add sample-level variables such as sample ID.

Usage

```
loadDataFromFileList(
    file_list, input_type,
    data_symbols = NULL,
    header = TRUE, sep = "")
```

```
combineSamples(
    file_list, input_type,
    data_symbols = NULL,
    header = TRUE, sep = "",
    seq_col,
    min_seq_length = NULL,
    drop_matches = NULL,
    subset_cols = NULL,
    sample_ids = NULL,
    subject_ids = NULL,
    group_ids = NULL)
```

Arguments

file_list	A character vector of file paths containing the data frames (one file per data frame).
input_type	A character string specifying the input file format; determines the function used to load each file. See details section below.
data_symbols	If input_type = "rda", a character vector specifying the name of each data frame (this can be of length 1 if all data frames have the same name).
header	Passed to read.table or read.csv if applicable.
sep	Passed to read.table or read.csv if applicable.
seq_col	Passed to filterInputData for each sample.
min_seq_length	Passed to filterInputData for each sample.
drop_matches	Passed to filterInputData for each sample.
subset_cols	Passed to filterInputData for each sample.
sample_ids	An optional character or numeric vector of sample IDs, whose length matches that of file_list.
subject_ids	An optional character or numeric vector of subject IDs, whose length matches that of file_list.
group_ids	An optional character or numeric vector of group IDs, whose length matches that of file_list.

Details

Valid options for input_type (and the corresponding function used to load each file) include "rds" (readRDS), "rda" (load), "csv" (read.csv) and "table" (read.table).

If input_type = "rda", the data_symbols argument specifies the name of each data frame in R. For example, given three R data frames named sample1, sample2 and sample3, with each data frame saved to a separate rda file using save()), we would pass c("sample1", "sample2", "sample3") to data_symbols.

For combineSamples, for each of sample_ids, subject_ids and group_ids that is non-null, a corresponding variable will be added to the combined data frame; these will be named SampleID, SubjectID and GroupID.

Value

A data frame containing the combined data rows from all files.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_samples <- file.path(data_dir, "samples")
dir.create(dir_samples, showWarnings = FALSE)
simulateToyData(sample_size = 5,
                output_dir = dir_samples,
                no_return = TRUE)


# Load data frames and combine
data <- loadDataFromFileList(
  file_list = list.files(dir_samples, full.names = TRUE),
  input_type = "rds")


## Same as above, but filter out seqs shorter than 3 characters,
## drop the count/freq/sampleID columns & add new sample ID column
data2 <- combineSamples(
  file_list = list.files(dir_samples, full.names = TRUE),
  input_type = "rds",
  seq_col = "CloneSeq",
  min_seq_length = 3,
  subset_cols = "CloneSeq",
  sample_ids = c("id01", "id02"))
```

---

encodeTCRSeqsByAtchleyFactor

Numerically Encode TCR Sequences Using Deep Learning

Description

Given a list of T-cell receptor CDR3 amino acid sequences, encodes each sequence as a 30-dimensional numeric vector based on the Atchley factor representation of its terms using a trained encoder.

Usage

```
encodeTCRSeqsByAtchleyFactor(
    cdr3_AA,
    contig_ids = seq_along(cdr3_AA))
```

Arguments

cdr3_AA	A character vector containing the TCR CDR3 amino acid sequences.
contig_ids	A numeric vector of the same length as cdr3_AA, used to uniquely identify its elements. By default, this is simply the sequence of integers from 1 to length(cdr3_AA).

Details

Encoding is performed using the BriseisEncoder python script. Each amino acid is represented by its five-dimensional vector of Atchley factors, which numerically encode its biological characteristics. Each TCR CDR3 amino acid sequence is embedded in Euclidean 30-space based on the Atchley factor representations of its constitutent terms, using a deep learning encoder and a pretrained model.

Value

A matrix with length(cdr3_AA) rows and 31 columns. Each row corresponds to a TCR sequence; the first column is contig_ids, while the remaining columns contain the 30-dimensional numerical encoding for each TCR sequence.

Note

The encoder was trained specifically on TCR CDR3 amino acid sequences and is not appropriate for use with other amino acid sequences.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
cdr3 <- c("CASSEAQGSGSTDTQYF",
          "CATTEGSNTGELFF",
          "CASSIGDNEQFF",
          "CATSRDPDRGQSDTQYF",
          "CASSPTGLSGNTIYF",
          "CASSEEAGKDTQYF",
          "CASSGGADTQYF",
          "CASSLGLATDTQYF",
          "CASSEKEEVGELFF",
          "CASSSRTSGGAGELFF")
encodeTCRSeqsByAtchleyFactor(cdr3)

## End(Not run)
```

`filterInputData` *Filter RepSeq Data Rows and/or Subset Columns*

Description

Filter data rows by sequence length and/or sequence content; subset data columns using a numeric or character vector of column references.

Usage

```
filterInputData(data, seq_col, min_seq_length = NULL,
                drop_matches = NULL,
                subset_cols = NULL,
                count_col = NULL)
```

Arguments

`data`	A data frame containing the immune repertoire sequencing data, with variables indexed by column and observations (e.g., clones or cells) indexed by row.
`seq_col`	The column name of `data` containing the receptor sequences for use in any filtering based on length or content.
`min_seq_length`	A numeric scalar, or `NULL`. Observations whose receptor sequences have fewer than `min_seq_length` characters are removed prior to network analysis.
`drop_matches`	An optional regular expression or character string. Observations whose receptor sequences return a match are removed prior to network analysis.
`subset_cols`	Optional vector of column names or numbers of `data`; if supplied, only these (in addition to the column referenced by `seq_col`) are included in the output. If `NULL` (the default), all columns of `data` are included.
`count_col`	Optional column name or number of `data`; if supplied, the specified column of the data will be coerced to numeric and data rows containing NA/NaNs in the count column will be dropped.

Details

The sequence column(s) of the data are coerced to character type and rows containing NA values in the sequence column(s) are dropped.

Value

A data frame containing the filtered rows and specified columns of `data`.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Generate some toy data for demonstration
raw_data <- simulateToyData()

# Filter data
filtered_data <-
  filterInputData(raw_data, seq_col = "CloneSeq",
                  min_seq_length = 13,
                  drop_matches = "[*|_]",
                  subset_cols = c("CloneSeq", "CloneFrequency", "SampleID"))
```

`findAssociatedClones` *Gather Clones in Public Neighborhoods Around Associated Sequences*

Description

Given multiple samples of RepSeq data from two groups and a list of receptor sequences associated to the comparison group (e.g., obtained using `findAssociatedSeqs`), this function does the following: for each associated sequence, from all samples that possess the associated sequence, gather all clones whose receptor sequences lie in a neighborhood of the associated sequence.

The public neighborhood for each associated sequence is stored in its own data frame and saved to its own file, in preparation for use with `buildAssociatedClusterNetwork`.

Usage

```
findAssociatedClones(

  ## Input ##
  file_list, input_type,
  data_symbols = NULL,
  header = TRUE, sep = "",
  sample_ids = 1:length(file_list),
  subject_ids = sample_ids,
  group_ids,
  seq_col,

  ## Search Criteria ##
  assoc_seqs,
  nbd_radius = 1,
  dist_type = "hamming",
  min_seq_length = 6,
  drop_matches = "[*|_]",

  ## Output ##
  subset_cols = NULL,
  output_dir =
    file.path(getwd(), "associated_neighborhoods"),
  output_type = "csv",
  verbose = FALSE
)
```

Arguments

file_list	Passed to loadDataFromFileList (one file per sample).
input_type	Passed to loadDataFromFileList.
data_symbols	Passed to loadDataFromFileList.
header	Passed to loadDataFromFileList.
sep	Passed to loadDataFromFileList.
sample_ids	A character or numeric vector of sample IDs, whose length matches that of file_list.
subject_ids	A character or numeric vector of subject IDs, whose length matches that of file_list. This can be disregarded if each sample comes from a distinct subject.
group_ids	A character or numeric vector of group IDs, which contains exactly two unique values and whose length matches that of file_list.
seq_col	The column name or number of each sample's data frame that contains the receptor sequences to associate with the comparison group.
assoc_seqs	A character vector containing the receptor sequences associated to the comparison group.
nbd_radius	The maximum distance between an element of assoc_seqs and other clones within its neighborhood.
dist_type	The distance metric used to measure the distance between sequences in nbd_radius. Valid options are "hamming" and "levenshtein".
min_seq_length	Passed to filterInputData() when loading each sample.
drop_matches	Passed to filterInputData().
subset_cols	Passed to filterInputData().
output_dir	The output directory; a valid output directory is required, since results are not returned in R.
output_type	A character string specifying the file format to use when writing each file. Valid options include "csv", "tsv", "rds" and "rda".
verbose	A logical scalar; if TRUE, additional console output will be printed reporting the number of clones in each sample belonging to the neighborhood for each associated sequence, as well as the total number of clones in each neighborhood across all samples.

Details

Each neighborhood's RepSeq data file is saved using the corresponding associated sequence for the filename, with the appropriate file extension appended based on the value of output_type (e.g., the neighborhood data for the sequence "CASSGAYEQYF" would be saved as "CASSGAYEQYF.csv" if output_type = "csv").

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

findAssociatedSeqs buildAssociatedClusterNetwork

Examples

```
## Not run:
## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)

samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## 1. Find Associated Sequences

# input files for step 1 (one per sample)
input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)

# group labels for the samples
group_labels <- c(rep("reference", samples / 2), rep("comparison", samples / 2))
```

```
# search across samples for associated sequences using Fisher's exact test
associated_seqs <- findAssociatedSeqs(
  file_list = input_files, input_type = "rds",
  group_ids = group_labels, groups = c("reference", "comparison"),
  min_seq_length = NULL, drop_matches = NULL,
  seq_col = "CloneSeq", outfile = NULL)
head(associated_seqs)


## 2. Find Associated Clones

# output directory for current step
dir_nbds <- file.path(data_dir, "assoc_seq_nbds")

# Identify neighborhood around each associated sequence
findAssociatedClones(
  file_list = input_files, input_type = "rds", group_ids = group_labels,
  seq_col = "CloneSeq", dist_type = "levenshtein",
  assoc_seqs = associated_seqs$ReceptorSeq,
  min_seq_length = NULL, drop_matches = NULL,
  output_dir = dir_nbds)

## End(Not run)
```

`findAssociatedSeqs` *Find Receptor Sequences Associated to a Comparison Group*

Description

Given multiple samples of RepSeq data from two groups, search for sequences associated to the comparison group based on Fisher's exact test P-value.

`findAssociatedSeqs` is for use with separate files and data frames per sample, while `findAssociatedSeqs2` is for use with a single data frame containing all samples.

Usage

```
findAssociatedSeqs(
  ## Input ##
  file_list, input_type,
  data_symbols = NULL,
  header = TRUE, sep = "",
  sample_ids = 1:length(file_list),
  subject_ids = sample_ids,
  group_ids, groups = c("group0", "group1"),
  seq_col, freq_col = NULL,

  ## Search Criteria ##
  min_seq_length = 7, drop_matches = "[*|_]",
  min_sample_membership = 5, pval_cutoff = 0.05,

  ## Output ##
  outfile = "associated_seqs.csv")
```

```
findAssociatedSeqs2(
  ## Input ##
  data, seq_col, sample_col,
  subject_col = sample_col,
  group_col, groups = c("group0", "group1"),
  freq_col = NULL,

  ## Search Criteria ##
  min_seq_length = 7, drop_matches = "[*|_]",
  min_sample_membership = 5, pval_cutoff = 0.05,

  ## Ouptut ##
  outfile = "associated_seqs.csv")
```

Arguments

file_list	Passed to loadDataFromFileList (one data frame per sample).
input_type	Passed to loadDataFromFileList.
data_symbols	Passed to loadDataFromFileList.
header	Passed to loadDataFromFileList.
sep	Passed to loadDataFromFileList.
sample_ids	A character or numeric vector of sample IDs, whose length matches that of file_list.
subject_ids	A character or numeric vector of subject IDs, whose length matches that of file_list. This can be disregarded if each sample comes from a distinct subject.
group_ids	A character or numeric vector of group IDs, which contains exactly two unique values and whose length matches that of file_list.
groups	A vector of length two having the same type as group_ids and containing the two unique values; the first and second elements identify the reference and comparison groups, respectively.
seq_col	The column name or number of each sample's data frame that contains the receptor sequences to associate with the comparison group.
freq_col	Optional. The column name or number of each sample's data frame that contains the clone frequency/clone fraction (i.e., measure of clonal abundance normalized based on the total count in the sample). This column must have the same name in each sample. If supplied, the maximum clone frequency of each associated sequence across all samples will be recorded in the returned output.
min_seq_length	Passed to filterInputData().
drop_matches	Passed to filterInputData().
min_sample_membership	Only sequences that appear in at least this many samples will be considered.
pval_cutoff	Only sequences with Fisher's exact test P-value below this cutoff will be returned.
outfile	A file path used to save the data frame containing the associated sequences (using write.csv).

data	A data frame containing the combined immune repertoire sequencing data for all samples, with variables indexed by column and observations indexed by row.
sample_col	The column of data containing the sample IDs.
subject_col	The column of data containing the subject IDs.
group_col	The column of data containing the group IDs.

Details

The data is first filtered by minimum sequence length and sequence content if applicable, then is filtered based on minimum sample membership, keeping only those receptor sequences that appear in at least min_sample_membership samples.

For each remaining sequence, Fisher's exact test is performed using stats::fisher.test(), based on the number of subjects possessing the sequence in each of the reference and comparison groups. Only sequences with a P-value below pval_cutoff will be retained. The resulting sequences are sorted by P-value in increasing order and returned along with some basic meta-data.

The returned ouput is intended for downstream use with findAssociatedClones and buildAssociatedClusterNetwor

Value

A data frame containing the following variables:

ReceptorSeq	The receptor sequences associated with the comparison group, based on the specified criteria.
shared_by_n_samples	The number of samples in which each sequence appears.
fisher_pvalue	The P-value on Fisher's exact test.
label	A character string containing meta-data in sentence formatting, for convenient use in, e.g., plot captions.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

findAssociatedClones buildAssociatedClusterNetwork

Examples

```
## Not run:
## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)
```

```
samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## 1. Find Associated Sequences

# input files for step 1 (one per sample)
input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)

# group labels for the samples
group_labels <- c(rep("reference", samples / 2), rep("comparison", samples / 2))

# search across samples for associated sequences using Fisher's exact test
associated_seqs <- findAssociatedSeqs(
  file_list = input_files, input_type = "rds",
  group_ids = group_labels, groups = c("reference", "comparison"),
  min_seq_length = NULL, drop_matches = NULL,
  seq_col = "CloneSeq", outfile = NULL)
head(associated_seqs)

## End(Not run)
```

`findPublicClusters` *Find Public Clusters Among RepSeq Samples*

Description

Given multiple samples of RepSeq data, perform network analysis on each sample individually and filter clusters by node count and clone count.

Usage

```
findPublicClusters(

  ## Input ##
  file_list, input_type,
  data_symbols = NULL, header = TRUE, sep = "",
  sample_ids = 1:length(file_list),
  seq_col, count_col = NULL,
  min_seq_length = 3, drop_matches = "[*|_]",

  ## Network ##
  top_n_clusters = 20,
  min_node_count = 10,
  min_clone_count = 100,

  ## Visualization ##
  plots = FALSE, print_plots = FALSE,
  plot_title = "auto", color_nodes_by = "cluster_id",

  ## Output ##
  output_dir = file.path(getwd(), "public_clusters"),
  output_type = "rds",
  output_dir_unfiltered = NULL,
  output_type_unfiltered = "rds",
  ...

)
```

Arguments

file_list	Passed to loadDataFromFileList(one data frame per sample).
input_type	Passed to loadDataFromFileList.
data_symbols	Passed to loadDataFromFileList.
header	Passed to loadDataFromFileList.
sep	Passed to loadDataFromFileList.
sample_ids	A character or numeric vector of sample IDs, whose length matches that of file_list. The values should be valid for use in filenames.
seq_col	The column name or number of each sample's data frame that contains the receptor sequences to be used as the basis of similarity between rows during network analysis. This column must have the same name in each sample.
count_col	The column name or number of each sample's data frame that contains the measure of clonal abundance to be used when filtering the clusters. This column must have the same name in each sample.
min_seq_length	Passed to buildRepSeqNetwork() when performing network analysis on each sample (prior to filtering the clusters).

drop_matches Passed to buildRepSeqNetwork().

top_n_clusters The top_n_clusters clusters with the highest node count will automatically be included among the public clusters.

min_node_count Clusters with node count of at least this value will be included among the public clusters.

min_clone_count
Clusters with aggregate clone count of at least this value will be included among the public clusters.

plots Passed to buildRepSeqNetwork() when performing network analysis on each sample (prior to filtering the clusters).

print_plots Passed to buildRepSeqNetwork().

plot_title Passed to buildRepSeqNetwork().

color_nodes_by Passed to buildRepSeqNetwork().

output_dir The outpout directory to save the filtered network results for each sample.

output_type A character string specifying the file format used to save the filtered results. Options include "rds", "csv" and "rda".

output_dir_unfiltered
The output directory to save the unfiltered network results for each sample, if desired. If NULL (the default), only the filtered results are saved.

output_type_unfiltered
A character string specifying the file format scheme used to save the unfiltered network results for each sample. Only applicable if output_dir_unfiltered is non-null. Passed to buildRepSeqNetwork().

... Other arguments to buildRepSeqNetwork(), not including node_stats, stats_to_include or cluster_stats (see details).

Details

All node-level and cluster-level network properties are automatically computed for each sample. The node-level properties are renamed to reflect their association to the sample-level network. Specifically, the properties are named ClusterIDInSample, SampleLevelNetworkDegree, SampleLevelTransitivity SampleLevelCloseness, SampleLevelCentralityByCloseness, SampleLevelCentralityByEigen, SampleLevelEigenCentrality, SampleLevelBetweenness, SampleLevelCentralityByBetweenness, SampleLevelAuthorityScore, SampleLevelCoreness, SampleLevelPageRank. A variable SampleID is added to both the node-level and cluster-level meta data for each sample.

After the clusters in each sample are filtered, the node-level and cluster-level meta data are saved in the subdirectories node_meta_data and cluster_meta_data, respectively, of output_dir; each is saved using the sample ID (the corresponding element of sample_id_list) as a filename prefix, followed by the appropriate file extension according to output_type.

The files containing the node-level meta data for the filtered clusters can be supplied to buildPublicClusterNetwork() in order to combine the public clusters across samples and perform network analysis. The files containing the cluster-level meta data for the filtered clusters can be supplied to buildPublicClusterNetworkByRepresenta to build a network using only a single representative sequence from each cluster.

The unfiltered network results for each sample can also be saved by supplying a directory to output_dir_unfiltered, if these results are desired for downstream analysis.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

buildPublicClusterNetwork buildPublicClusterNetworkByRepresentative

Examples

```
## Not run:

## Generate some toy data for demonstration

# Use temp dir
data_dir <- tempdir()

# Directory to store input files
dir_input_samples <- file.path(data_dir, "input_samples")
dir.create(dir_input_samples, showWarnings = FALSE)

samples <- 30
affixes <- c("AAAA", "AASA", "AACA", "AAQA", "AAQ", "AAA", "AASAA", "AAAAA")
affix_probs_g0 <- rep(1 / length(affixes),
                      times = length(affixes) * samples / 2)
affix_probs_g1 <- rep(c(1, 5, 1, 1, 1, 1, 5, 1), times = samples / 2)
affix_probs <- matrix(c(affix_probs_g0, affix_probs_g1),
                      nrow = samples, byrow = TRUE)
new_probs_g0 <- rep(c(1/2, 1/6, 1/6, 1/6), times = samples / 2)
new_probs_g1 <- rep(c(1/3, 1/6, 1/6, 1/3), times = samples / 2)
new_probs <- matrix(c(new_probs_g0, new_probs_g1),
                    nrow = samples, byrow = TRUE)
simulateToyData(
  samples = samples,
  sample_size = 30,
  prefix_length = 1,
  prefix_chars = c("A", "C"),
  prefix_probs = cbind(rep(1, samples), rep(0, samples)),
  affixes = affixes,
  affix_probs = affix_probs,
  num_edits = 4,
  edit_pos_probs = function(seq_length) {
    dnorm(seq(-4, 4, length.out = seq_length))
  },
  edit_ops = c("insertion", "deletion", "transmutation"),
  edit_probs = c(5, 1, 4),
  new_chars = c("A", "S", "C", "Q"),
  new_probs = new_probs,
  output_dir = dir_input_samples,
  no_return = TRUE
)


## Find Public Clusters in Each Sample

input_files <- file.path(dir_input_samples, paste0("Sample", 1:samples, ".rds"))
head(input_files)
```

```
dir_filtered_samples <- file.path(data_dir, "filtered_samples")
findPublicClusters(
  file_list = input_files, input_type = "rds",
  sample_ids = paste0("Sample", 1:samples),
  seq_col = "CloneSeq", count_col = "CloneCount",
  min_seq_length = NULL, drop_matches = NULL,
  output_dir = dir_filtered_samples)


## End(Not run)
```

`generateNetworkFromAdjacencyMat` *Generate the Network igraph for an Adjacency Matrix*

Description

A simple wrapper function used for convenience. Given an adjacency matrix for an undirected network, generates the `igraph` object containing the undirected network graph corresponding to the adjacency matrix after simplifying and removing loops.

Usage

```
generateNetworkFromAdjacencyMat(adjacency_matrix)
```

Arguments

`adjacency_matrix`
: Passed to `igraph::graph_from_adjacency_matrix`. A square adjacency matrix. Sparse matrices created using the `Matrix` package are supported.

Value

An `igraph` graph object.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData(sample_size = 10)

adjmat <- sparseAdjacencyMatFromSeqs(toy_data$CloneSeq)
net <- generateNetworkFromAdjacencyMat(adjmat)
```

generateNetworkGraphPlots
Generate Multiple Plots of a Network

Description

Generate multiple plots using an igraph and corresponding node-level meta data.

Usage

```
generateNetworkGraphPlots(
  igraph, data, print_plots = TRUE,
  plot_title = NULL, plot_subtitle = NULL,
  color_nodes_by = NULL, color_scheme = "default",
  color_legend = "auto", color_title = "auto",
  edge_width = 0.1, size_nodes_by = 0.5,
  node_size_limits = NULL, size_title = "auto")
```

Arguments

igraph	An igraph object containing the edge list for the network.
data	A data frame containing the node-level meta-data for the network, with each row corresponding to a node.
print_plots	A logical scalar; should plots be printed in the R plotting window?
plot_title	Passed to plotNetworkGraph().
plot_subtitle	Passed to plotNetworkGraph().
color_nodes_by	Optional. Column name or number of data used to color network nodes. If a vector of values is supplied, one plot will be generated for each value.
color_scheme	Passed to plotNetworkGraph. If a vector is supplied to color_nodes_by, this argument will optionally accept a vector of matching length; if supplied, corresponding values of color_scheme and color_nodes_by will be used together in each call to plotNetworkGraph.
color_legend	Passed to plotNetworkGraph().
color_title	Passed to plotNetworkGraph. If a vector is supplied to color_nodes_by, this argument will optionally accept a vector of matching length; if supplied, corresponding values of color_title and color_nodes_by will be used together in each call to plotNetworkGraph.
edge_width	Passed to plotNetworkGraph.
size_nodes_by	Numeric value specifying the node size, or a column name or number of data for dynamically sizing the nodes.
node_size_limits	Passed to plotNetworkGraph().
size_title	Passed to plotNetworkGraph().

Details

This function's primary use is to generate multiple plots of a single network, each using a different variable to color-code the nodes.

Value

A list containing one item for each plot generated. Each plot is stored as a ggraph object.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

plotNetworkGraph

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Plot network graph
net_plot <- generateNetworkGraphPlots(
  net$igraph, net$node_data, color_nodes_by = c("SampleID", "CloneCount"))

## End(Not run)
```

---

generateNetworkObjects

Generate Core Objects for a RepSeq Network

Description

Given immune repertoire sequencing data, returns the adjacency matrix, igraph and node meta data for the repetroire network based on sequence similarity.

Usage

```
generateNetworkObjects(data, seq_col,
                       dist_type = "hamming",
                       dist_cutoff = 1,
                       drop_isolated_nodes = TRUE)
```

Arguments

data	A data frame containing the RepSeq data, with clones/cells indexed by row.
seq_col	The column name or number of data containing the receptor sequences to be used as the basis of similarity between rows. Also accepts a vector of length 2 specifying distinct sequence columns (e.g., alpha, beta chains); then two rows are similar only if both types of sequences are similar.
dist_type	The type of function to use as a measure of similarity between two receptor sequences. Valid options are "hamming" (the default), "levenshtein" and "euclidean_on_atchley" (only applicable to TCR CDR3 amino acid sequences).
dist_cutoff	A nonnegative scalar specifying the maximum distance threshold for similarity between receptor sequences.
drop_isolated_nodes	A logical scalar; should observations whose receptor sequences are not similar to any other sequences be dropped from the network?

Details

This function is essentially a lightweight version of buildRepSeqNetwork, omitting the input filtering, computation of network properties, generation of plots and saving of output.

Value

If the constructed network contains no edges, the function will return NULL with a warning. Otherwise, a list containing the following items:

igraph	An igraph object containing the edge list for the network.
adjacency_matrix	The network graph adjacency matrix, stored as a sparse matrix of class dgCMatrix from the Matrix package.
node_data	A data frame containing the node-level meta-data for the network. This data frame contains all column names of data. The data frame will contain one row for each node that remains in the network after filtering based on drop_isolated_nodes, if applicable. The row names of the original input data will be preserved.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

## End(Not run)
```

getClusterStats *Compute Cluster-Level Network Statistics*

Description

Given immune repertoire sequence data and the adjacency matrix for the corresponding repertoire network, perform clustering and compute cluster-level network statistics.

Usage

```
getClusterStats(data,
                adjacency_matrix,
                seq_col = NULL,
                count_col = NULL,
                cluster_id_col = NULL,
                degree_col = NULL,
                cluster_fun = cluster_fast_greedy)
```

Arguments

data	A data frame containing the immune repertoire sequence data, with clones/cells indexed by row.
adjacency_matrix	A square adjacency matrix corresponding to the repertoire network for the rows in data, with row dimension matching that of data. Sparse matrices created using the Matrix package are supported.
seq_col	Optional. The column name or number of data containing the receptor sequences (e.g., TCR CDR3 nucleotide sequence or amino acid sequence). If supplied, then related cluster-level properties will be computed.
count_col	Optional. The column name or number of data containing the counts (i.e., clone count or UMI count). If supplied, then related cluster-level properties will be computed.
cluster_id_col	The column name or number of data containing the cluster membership for each node. If NULL, clustering will be performed using igraph::cluster_fast_greedy() in order to determine cluster membership.
degree_col	The column name or number of data containing the network degree for each node. If NULL, the network degree will be computed using igraph::degree.
cluster_fun	Passed to addClusterMembership. Controls the algorithm used for cluster identification.

Details

The cluster-level statistics diameter_length, assortativity, transitivity, edge_density, degree_centrality_index, closeness_centrality_index, eigen_centrality_index, and eigen_centrality_ei are computed for each cluster by first generating the network graph for the cluster using the adjacency submatrix corresponding to that cluster. As a result, computing cluster statistics can potentially take some time for very large networks with many large clusters.

Value

A data frame containing one row for each cluster and the following columns:

`cluster_id` The cluster ID number.

`node_count` The number of nodes in the cluster.

`mean_seq_length`
The mean sequence length in the cluster.

`mean_degree` The mean network degree in the cluster.

`max_degree` The maximum network degree in the cluster.

`seq_w_max_degree`
The receptor sequence possessing the maximum degree within the cluster.

`agg_count` The aggregate count among all nodes in the cluster (based on the counts in `count_col`).

`max_count` The maximum count among all nodes in the cluster (based on the counts in `count_col`).

`seq_w_max_count`
The receptor sequence possessing the maximum count within the cluster.

`diameter_length`
The longest geodesic distance in the cluster, computed using `length(igraph::get_diameter())`.

`assortativity` The assortativity coefficient of the cluster's graph, based on the degree (minus one) of each node in the cluster (with the degree computed based only upon the nodes within the cluster). Computed using `igraph::assortativity_degree`.

`global_transitivity`
The transitivity (i.e., clustering coefficient) for the cluster's graph, which estimates the probability that adjacent vertices are connected. Computed using `igraph::transitivty()` with `type = "global"`.

`edge_density` The number of edges in the cluster as a fraction of the maximum possible number of edges. Computed using `igraph::edge_density()`.

`degree_centrality_index`
The cluster-level centrality index based on degree within the cluster graph. Computed using `igraph::centr_degree()$centralization`.

`closeness_centrality_index`
The cluster-level centrality index based on closeness, i.e., distance to other nodes in the cluster. Computed using `igraph::centr_clo()$centralization`.

`eigen_centrality_index`
The cluster-level centrality index based on the eigenvector centrality scores, i.e., values of the first eigenvector of the adjacency matrix for the cluster. Computed using `igraph::centr_eigen()$centralization`.

`eigen_centrality_eigenvalue`
The eigenvalue corresponding to the first eigenvector of the adjacency matrix for the cluster. Computed using `igraph::eigen_centrality()$value`.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Get cluster stats
cluster_info <-
  getClusterStats(net$node_data, net$adjacency_matrix,
                  seq_col = "CloneSeq", count_col = "CloneCount")

## End(Not run)
```

`getNeighborhood` *Compute the Neighborhood Around a Cell/Clone*

Description

Given immune repertoire sequence data and a target receptor sequence, return the subset of the data consiting of those cells/clones whose receptor sequences are within a specified distance of the target.

Usage

```
getNeighborhood(data, seq_col, target_seq,
                dist_type = "hamming", max_dist = 1)
```

Arguments

`data`	Data frame containing the RepSeq data.
`seq_col`	The column name or number of `data` containing the receptor sequences of the same type as `target_seq`.
`target_seq`	A character string containing the receptor sequence of the cell/clone around which to form a neighborhood.
`dist_type`	A character string specifying the distance type on which the neighborhood is based; valid options are `"hamming"` and `"levenshtein"`.
`max_dist`	A numeric scalar specifying the neighborhood radius. The neighborhood consists of all clones whose receptor sequences are at most `max_dist` from the target sequence.

Details

The neighborhood will only be computed if `target_seq` is present in the data, i.e., in the column of `data` referenced by `seq_col`.

Value

If `target_seq` is present in the data, a data frame containing the subset corresponding to the neighborhood; otherwise, the function returns `NULL`.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData(sample_size = 500)

# Get neighborhood around first clone sequence
nbd <- getNeighborhood(toy_data, seq_col = "CloneSeq",
                       target_seq = toy_data$CloneSeq[[1]])
```

---

hamDistBounded *Bounded Computation of Hamming Distance*

Description

Computes the Hamming distance between two strings subject to a specified upper bound.

Usage

```
hamDistBounded(a, b, k)
```

Arguments

a	A character string.
b	A character string to be compared to a.
k	The upper bound on the Hamming distance between a and b.

Details

The Hamming distance measures the number of single-character transformations required to transform one string into another. It is defined only for strings of equal length. If a and b differ in length, placeholder characters are appended to the shorter string until the lengths match; each placeholder character in the shorter string is treated as non-matching with the character in the corresponding position of the longer string.

Computation is halted if the Hamming distance is determined to exceed the upper bound k; this reduces the computation required when distinguishing between values above the upper bound is unnecessary.

Value

An integer whose value is the Hamming distance between a and b if this distance is at most k; otherwise the value is -1.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
hamDistBounded("foo", "bar", 1)

hamDistBounded("foo", "bar", 10)
```

`installPythonModules` *Install Python Modules Used by NAIR*

Description

Checks whether the Python modules used by the `NAIR` package are available, and automatically installs any that are missing via `reticulate::py_install()`.

Usage

```
installPythonModules(method = "auto", conda = "auto", pip = FALSE)
```

Arguments

`method`	Passed to the `method` argument of `reticulate::py_install()`. See `?reticulate::py_install()` for details.
`conda`	Passed to the `conda` argument of `reticulate::py_install()`. See `?reticulate::py_install()` for details.
`pip`	Passed to the `pip` argument of `reticulate::py_install()`. See `?reticulate::py_install()` for details. For the `levenshtein` Python module, the call to `reticulate::py_install()` always uses `pip = TRUE`.

Details

The modules checked for and installed are `numpy`, `pandas`, `tensorflow` and `keras`. Calling this function is equivalent to making individual calls to `recitulate::py_install()` for each of these modules.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

kmeansAtchley *Analyze TCR Samples Using Numerical Encoder and K-Means Clustering*

Description

Given samples of TCR RepSeq data, convert the CDR3 amino acid sequences into numeric vectors using deep learning, perform K-means clustering, profile how each sample's unique TCR sequences are distributed among the clusters, and compare these profiles across samples.

Usage

```
kmeansAtchley(
  data,
  amino_col = "AminoAcidSeq",
  sample_col = "SampleID",
  group_col, k = 100,
  pdf_width = 15,
  pdf_height = 15,
  margin_cluster_heatmap = 25,
  margin_corr_heatmap = 15,
  use_viridis = FALSE,
  output_dir = getwd(),
  file_cluster_heatmap =
    "atchley_kmeans_cluster_relative_size_profiles_by_sample.pdf",
  file_corr_heatmap =
    "atchley_kmeans_corr_in_cluster_size_profile_between_samples.pdf",
  return_output = FALSE)
```

Arguments

data	A data frame containing the combined immune repertoire sequencing data for all samples, with variables indexed by column and observations indexed by row.
amino_col	The column of data containing the CDR3 amino acid sequences.
sample_col	The column of data containing the sample IDs.
group_col	The column of data containing the group IDs.
k	The number of clusters for K-means clustering.
pdf_width	Passed to the width argument of grDevices::pdf().
pdf_height	Passed to the height argument of grDevices::pdf().
margin_cluster_heatmap	The margin width of column names and row names in the heatmap for each sample's cluster profile.
margin_corr_heatmap	The margin width of column names and row names in the heatmap for the correlation in cluster profiles between samples.
use_viridis	A logical scalar indicating whether to use the cividis color scale from the viridis package instead of the RdBu color scale from RColorBrewer. This yields more readable results when printing in black-and-white, and is more robust against color vision deficiency.

`output_dir` The directory in which to save pdf files of the two heatmaps produced.

`file_cluster_heatmap`
The pdf filename for the heatmap for each sample's cluster profile.

`file_corr_heatmap`
The pdf filename for the heatmap for correlation in cluster profiles between samples.

`return_output` A logical scalar indicating whether to return the numeric vectors encoding the TCR sequences, the K-means cluster membership table, and each sample's profile of TCR representation across clusters.

Details

Each unique TCR sequence is encoded as a 30-dimensional numeric vector using `encodeTCRSeqsByAtchleyFactor()`, and K-means clustering is performed on the encoded TCR sequences from all samples, resulting in `k` clusters.

For each sample, its TCR representation profile across the `k` clusters is computed as a length-`k` vector whose ith element is the fraction of the sample's unique TCR sequences that belong to the ith cluster.

Two heatmaps are generated and saved to file: the first shows the TCR representation profile values across samples and clusters; the second shows the correlation in TCR represenation profiles between pairs of samples.

Value

If `return_output = TRUE`, then a list containing the following items:

`kmeans_cluster_ids`
A data frame with two variables, `cdr3` and `kmeanClusterID`, containing the unique TCR sequences and the cluster to which each belongs.

`encoded_values` A matrix returned by `encodeTCRSeqsByAtchleyFactor` containing the numerically encoded values of the TCR sequences.

`cluster_TCR_profiles`
A matrix with `k` rows and one column per sample; each column is the TCR representation profile for a sample, with each value recording the fraction of that sample's unique TCR sequences belonging to the cluster for the corresponding row (thus each column sums to 1).

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
toy_data <- simulateToyData(
  samples = 20,
  sample_size = 50,
  prefix_length = 0,
  prefix_chars = "",
```

```
  prefix_probs = matrix(1, nrow = 20),
  affixes = c("CASSLGYEQYF", "CASSLGETQYF",
              "CASSLGTDTQYF", "CASSLGTEAFF",
              "CASSLGGTEAFF", "CAGLGGRDQETQYF",
              "CASSQETQYF", "CASSLTDTQYF",
              "CANYGYTF", "CANTGELFF",
              "CSANYGYTF"),
  affix_probs = matrix(1, ncol = 11, nrow = 20),
)
toy_data$GroupID <- rep(c("G1", "G2"), each = 500)

atchley <- kmeansAtchley(
  data = toy_data,
  k = 3,
  amino_col = "CloneSeq",
  sample_col = "SampleID",
  group_col = "GroupID",
  output_dir = tempdir(),
  return_output = TRUE)

## End(Not run)
```

`levDistBounded` *Bounded Computation of Levenshtein Distance*

Description

Computes the Levenshtein distance between two strings subject to a specified upper bound.

Usage

```
levDistBounded(a, b, k)
```

Arguments

a	A character string.
b	A character string to be compared to a.
k	The upper bound on the Levenshtein distance between a and b.

Details

The Levenshtein (edit) distance measures the minimum number of single-character edits (which include insertions, deletions and transformations) required to change one string into another.

Computation is halted if the Levenshtein distance is determined to exceed the upper bound k; this reduces the computation required when distinguishing between values above the upper bound is unnecessary.

Value

An integer whose value is the Levenshtein distance between a and b if this distance is at most k; otherwise the value is -1.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
levDistBounded("foo", "bar", 1)
levDistBounded("foo", "bar", 10)
levDistBounded("foobar", "fubar", 10)
```

---

plotNetworkGraph *Plot the Graph of an Immune Repertoire Network*

Description

A wrapper function for customized calling of functions from the ggraph package. Given an igraph network object, generates a ggraph according to the user specifications.

Usage

```
plotNetworkGraph(igraph,
                 plot_title = NULL,
                 plot_subtitle = NULL,
                 color_nodes_by = NULL,
                 color_scheme = "default",
                 color_legend = "auto",
                 color_title = "auto",
                 edge_width = 0.1,
                 size_nodes_by = 0.5,
                 node_size_limits = NULL,
                 size_title = "auto",
                 outfile = NULL)
```

Arguments

igraph	An igraph object containing the network graph to be plotted.
plot_title	A character string or NULL. The plot title, to be passed to ggplot2::labs.
plot_subtitle	A character string or NULL. The plot subtitle, to be passed to ggplot2::labs.
color_nodes_by	Passed to the color aesthetic mapping of ggraph::geom_node_point, used to encode the color of each node. Either a vector whose length matches the number of nodes in network, or NULL. If the vector is numeric and contains non-integer values, a continuous color scale will be used; otherwise, a discrete color scale will be used.
color_scheme	A character string specifying the color palette used to color the nodes: either "default" for default ggplot2 colors, a viridis color map option (e.g., "viridis", "plasma", etc., or one the corresponding strings "A" through "H"; see ?viridis for details), or a palette from grDevices::hcl.pals() (these can only be used with discrete color scales).

color_legend Accepts a logical scalar specifying whether to display the color legend on the plot. The default value of "auto" shows the color legend if color_nodes_by is a continuous variable or a discrete variable with at most 20 distinct values.

color_title A character string or NULL specifying the title for the color legend. If "auto" and a vector was supplied for color_nodes_by, the color legend title will attempt to use the name of the supplied vector.

edge_width A numeric scalar specifying the width of the graph edges in the plot. Passed to the width argument of ggraph::geom_edge_link0.

size_nodes_by Passed to the size aesthetic mapping of ggraph::geom_node_point, used to encode the size of each node. Either a numeric vector with positive entires whose length matches the number of nodes in network, a positive-valued number (denoting a fixed constant size for each node), or NULL.

size_title A character string or NULL specifying the title for the size legend. The size legend is only displayed if a vector was supplied for size_nodes_by, in which case size_title = "auto" will attempt to use the name of the supplied vector.

node_size_limits A numeric vector of length 2 with positive entries, or NULL. If a vector of values is provided, the node sizes will be rescaled using the first value for the minimum node size and the second value for the maximum node sizes. Only applicable if a vector is supplied for size_nodes_by.

outfile A character string containing a valid file name ending in ".pdf", or NULL. If a file name is supplied, the plot will be saved to the file.

Value

A ggraph object.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Plot network graph
net_plot <- plotNetworkGraph(
  net$igraph,
  color_nodes_by = net$node_data$SampleID,
  color_scheme = "viridis",
  size_nodes_by = net$node_data$CloneCount,
  node_size_limits = c(0.5, 3))

print(net_plot)
```

```
## End(Not run)
```

---

saveNetwork *Save List of RepSeq Network Objects*

Description

Given a list conforming to the output of `buildRepSeqNetwork()`, saves its contents according to the specified file format scheme. Also saves any graph plots present to a single pdf.

Usage

```
saveNetwork(net, output_dir = getwd(),
            output_type = "individual",
            output_filename = "MyRepSeqNetwork",
            pdf_width = 12, pdf_height = 10)
```

Arguments

`net`	A list conforming to the format of the value returned by `buildRepSeqNetwork`. Must contain at least the components `igraph` (an igraph object), `adjacency_matrix` (a matrix or dgCMatrix) and `node_data` (a data frame); the components `cluster_data` (a data frame) and `plots` (a list of `ggraph` objects) are optional, and will also be saved if present.
`output_dir`	The directory in which to save the file(s).
`output_type`	A character string specifying the file format to use when writing output to file. Default `"individual"` saves each item as a separate, uncompressed file, with data frames saved in csv format. `"rda"` and `"rds"` save the output list as a rda and rds file, respectively. For all output types, any plots present will also be saved in a pdf file.
`output_filename`	A character string to be used as a common filename prefix for any files saved.
`pdf_width`	Passed to the `width` argument of `grDevices::pdf()`.
`pdf_height`	Passed to the `height` argument of `grDevices::pdf()`.

Details

When `output_type = "individual"`, the igraph is saved in edgelist format as a .txt file; the adjacency matrix is saved either in .csv format (for dense matrices) or .mtx format (for sparse matrices); and plots are saved only in a .pdf file, meaning that any plots will need to be generated again in order to be modified. If `output_type` is `"rds"` or `"rda"`, the entire list `net` will be saved (the R symbol for the object will be `net` if saving as .rda).

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

saveNetwork(net, output_dir = tempdir())
```

---

saveNetworkPlots *Save List of GGPlots to a Single PDF*

Description

Print a list of ggplot objects to a single pdf file.

Usage

```
saveNetworkPlots(plotlist,
                 outfile = "MyRepSeqNetwork.pdf",
                 pdf_width = 12, pdf_height = 10)
```

Arguments

plotlist	A list of ggplot objects.
outfile	The file to which the printed plots should be written.
pdf_width	Passed to the width argument of grDevices::pdf().
pdf_height	Passed to the height argument of grDevices::pdf().

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

See Also

plotNetworkGraph

Examples

```
# Simulate some toy data for demonstration
toy_data <- simulateToyData()

# Generate network for data
net <- generateNetworkObjects(toy_data, "CloneSeq")

# Plot network graph
net$plots <- generateNetworkGraphPlots(
  net$igraph, net$node_data, color_nodes_by = c("SampleID", "CloneCount"))

# Save plots
saveNetworkPlots(net$plots, outfile = file.path(tempdir(), "network.pdf"))
```

`simulateToyData` *Generate Toy RepSeq Data*

Description

Generates toy data that can be used to test or demonstrate the behavior of functions in the `NAIR` package. Created as a lightweight tool for use in tests, examples and vignettes. This function is not intended to simulate realistic data.

Usage

```
simulateToyData(
    samples = 2,
    chains = 1,
    sample_size = 100,
    prefix_length = 7,
    prefix_chars = c("G", "A", "T", "C"),
    prefix_probs = rbind(
      "sample1" = c(12, 4, 1, 1),
      "sample2" = c(4, 12, 1, 1)),
    affixes = c("AATTGG", "AATCGG", "AATTCG",
                "AATTGC", "AATTG", "AATTC"),
    affix_probs = rbind(
      "sample1" = c(10, 4, 2, 2, 1, 1),
      "sample2" = c(1, 1, 1, 2, 2.5, 2.5)),
    num_edits = 0,
    edit_pos_probs = function(seq_length) {
      stats::dnorm(seq(-4, 4, length.out = seq_length))
    },
    edit_ops = c("insertion", "deletion", "transmutation"),
    edit_probs = c(5, 1, 4),
    new_chars = prefix_chars,
    new_probs = prefix_probs,
    output_dir = NULL,
    no_return = FALSE,
    seed_value = 42
)
```

Arguments

samples	The number of distinct samples to include in the data.
chains	The number of chains (either 1 or 2) for which to generate receptor sequences.
sample_size	The number of observations to generate per sample.
prefix_length	The length of the random prefix generated for each observed sequence. Specifically, the number of elements of prefix_chars that are sampled with replacement and concatenated to form each prefix.
prefix_chars	A character vector containing characters or strings from which to sample when generating the prefix for each observed sequence.
prefix_probs	A numeric matrix whose column dimension matches the length of prefix_chars and with row dimension matching the value of samples. The ith row specifies the relative probability weights assigned to each element of prefix_chars when sampling to form the prefix for each sequence in the ith sample.
affixes	A character vector containing characters or strings from which to sample when generating the suffix for each observed sequence.
affix_probs	A numeric matrix whose column dimension matches the length of affixes and with row dimension matching the value of samples. The ith row specifies the relative probability weights assigned to each element of affixes when sampling to form the suffix for each sequence in the ith sample.
num_edits	A nonnegative integer specifying the number of random edit operations to perform on each observed sequence after its initial generation.
edit_pos_probs	A function that accepts a nonnegative integer (the character length of a sequence) as its argument and returns a vector of this length containing probability weights. Each time an edit operation is performed on a sequence, the character position at which to perform the operation is randomly determined according to the probabilities given by this function.
edit_ops	A character vector specifying the possible operations that can be performed for each edit. The default value includes all valid operations (insertion, deletion, transmutation).
edit_probs	A numeric vector of the same length as edit_ops, specifying the relative probability weights assigned to each edit operation.
new_chars	A character vector containing characters or strings from which to sample when performing an insertion edit operation.
new_probs	A numeric vector of the same length as new_chars, specifying the relative probability weights assigned to each possible character or string to be inserted.
output_dir	An optional character string specifying a file directory to save the generated data. One file will be generated per sample.
no_return	A logical flag that can be used to prevent the function from returning the generated data. If TRUE, the function will instead return TRUE once all processes are complete.
seed_value	Passed to set_seed at the start of function execution.

Details

Each observed sequence is obtained by separately generating a prefix and suffix according to the specified settings, then joining the two and performing sequential rounds of edit operations randomized according to the user's specifications.

Count data is generated for each observation; note that this count data is generated independently from the observed sequences and has no relationship to them.

Value

If `no_return = FALSE` (the default), A data frame whose contents depend on the value of the `chains` argument.

For `chains = 1`, the data frame contains the following variables:

`CloneSeq`	The "receptor sequence" for each observation.
`CloneFrequency`	The "clone frequency" for each observation (clone count as a proportion of the aggregate clone count within each sample).
`CloneCount`	The "clone count" for each observation.
`SampleID`	The sample ID for each observation.

For `chains = 2`, the data frame contains the following variables:

`AlphaSeq`	The "alpha chain" receptor sequence for each observation.
`AlphaSeq`	The "beta chain" receptor sequence for each observation.
`UMIs`	A "unique count" for each observation.
`Count`	The "count" for each observation.
`SampleID`	The sample ID for each observation.

If `no_return = FALSE`, the function returns `TRUE` upon completion.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
## Not run:
dat1 <- simulateToyData()

dat2 <- simulateToyData(chains = 2)

simulateToyData(sample_size = 500,
                num_edits = 10,
                seed_value = 1,
                no_return = TRUE,
                output_dir = tempdir())

dat4 <-
  simulateToyData(
    samples = 5,
    sample_size = 50,
    prefix_length = 0,
    prefix_chars = "",
    prefix_probs = matrix(1, nrow = 5),
    affixes = c("CASSLGYEQYF", "CASSLGETQYF",
                "CASSLGTDTQYF", "CASSLGTEAFF",
                "CASSLGGTEAFF", "CAGLGGRDQETQYF",
                "CASSQETQYF", "CASSLTDTQYF",
```

```
                "CANYGYTF", "CANTGELFF",
                "CSANYGYTF"),
    affix_probs = matrix(1, ncol = 11, nrow = 5),
  )

## End(Not run)
```

`sparseAdjacencyMatFromSeqs` *Sparse Adjacency Matrix for Immune Repertoire Network by Clone Sequence Similarity*

Description

Given a list of T-cell or B-cell receptor sequences, generates the adjacency matrix (in sparse format) for the corresponding network graph based on either the Hamming or Levenshtein distance using a specified cutoff.

Usage

```
sparseAdjacencyMatFromSeqs(
    seqs,
    dist_type = "hamming",
    max_dist = 1,
    drop_isolated_nodes = TRUE)
```

Arguments

`seqs`	A character vector containing the TCR/BCR sequences.
`dist_type`	A character string specifying the distance type; valid options are `"hamming"` and `"levenshtein"`.
`max_dist`	A positive integer specifying the maximum distance at which two TCR/BCR sequences are concidered adjacent (joined by an edge) in the network graph.
`drop_isolated_nodes`	A logical scalar; if `TRUE`, sequences with no adjacencies will have their rows/columns excluded from the matrix.

Details

The adjacency matrix of a graph with n nodes/vertices is the symmetric $n \times n$ matrix for which the (i, j)th entry is equal to 1 if nodes i and j are adjacent (joined by an edge in the graph) and 0 otherwise.

Each element of `seqs` represents a node in the corresponding network graph; two nodes are adjacent if the distance between their sequences, as measured by `dist_type`, is at most `max_dist`.

Value

A sparse matrix of class `dgCMatrix` from the `Matrix` package.

If `drop_isolated_nodes = TRUE`, the element of `seqs` that corresponds to each row (equivalently, column) is specified in the `Dimnames` of this matrix. The `Dimnames` can be accessed using `dimnames()`. This returns a list with two elements, each a character vector whose length matches the row dimension of the matrix. These respectively specify the indices and the elements of `seqs` corresponding to each row.

Author(s)

Brian Neal, Hai Yang, Jason Cham, Zenghua Fan, Tao He and Li Zhang.

References

https://github.com/mlizhangx/Network-Analysis-for-Repertoire-Sequencing-

Examples

```
sparseAdjacencyMatFromSeqs(
    c("fee", "fie", "foe", "fum", "foo"))
```

Index