Package 'LINEAGE'

1 Introduction

Lineage analysis based on scRNA-seq could provide key insights into the fate of individual cells and inform therapeutic intervention for diseases. However, such analysis is limited by several technical challenges. On top of considerable computational resources, these analyses also require specific types of matching data. Therefore, a computational algorithm for label-free lineage analysis based on endogenous markers is desired.

We chose mitochondrial RNA variant as endogenous makers for lineage analysis in consideration of relatively small mitochondrial genome, mitochondrial genome polymorphism and heritable mitochondrial variation. In order to select the clonal feature without pre-existing knowledge, we defined “cross entropy” as the cell distribution similarity among subspaces. These subspaces are obtained by hierarchical clustering according to the variant frequency dynamic patterns. Then we designed a lineage analysis algorithm called Label-free IdeNtification of Endogenous informAtive sinGle cell mitochondrial RNA mutation for lineage analysis and clonal Evolution (LINEAGE) by integrating the low cross-entropy subspaces identification with a consensus clustering method.

We have provided an example data and its validated cell lineage information in “TF1_clones.rda” at
https://github.com/songjiajia2018/LINEAGE/data , which might be helpful for users to go through this manual.

2 Citation

Please cite the following articles when using *LINEAGE*.

3 Install

There are 2 options to install LINEAGE.

- Option 1: using the devtools package to install directly from GitHub

```
# install.packages("devtools")
devtools::install_github("songjiajia2018/LINEAGE")
```

- Option 2: download the source code from github, and install as follow

The source code package is provided at
https://github.com/songjiajia2018/LINEAGE/source_pkg

```
install.packages("the_directory_that_contain_the_package/LINEAGE_0.99.1
.tar.gz",repos=NULL,type="source")
```

4 Example data

An example data is available in the source codes.

TF1_clones.rda is a list containing the input mitochondrial genotype matrix (TF1_clones$data) and the experimental validated cell lineage information (TF1_clones$rlabel).

5 Functions

5.1 lineage: perform LINEAGE

Description

Using mitochondrial genotype matrix as input, select the informative variants of mitochondrial RNA for clones and perform lineage analysis with a consensus clustering method.

Usage

```
lineage(
  data,
  repeats = 30,
  thread = 10
)
```

Arguments

data	A dataframe containing mitochondrial variants frequency matrix, where a column represented a single cell and a row represented variants frequency of a specific mitochondrial genotype, and two other column "altAllele" and "refAllele". "altAllele" column represents the mutant allele; "refAllele" column represents the reference allele. Required.
repeats	Number of iterations. Default:30.
thread	Number of threads to use. NULL for non-parallel iterative optimization. Default:10.

Value

Returns a Rdata containing results of all repeats.

Examples

```
data("TF1_clones")
data=TF1_clones$data
## performs a non-parallel iteration process
result=lineage(data, repeats=30, thread=NULL)
## performs a parallel iteration process
result=lineage(data, repeats=30, thread=10)
```

5.2 data_prepare: matrix splitting based on mutation types

Description

Split the mitochondrial genotype matrix into 12 submatrices according to mutation types.

Usage

```
data_prepare(data)
```

Arguments

data	A dataframe containing mitochondrial variants frequency matrix and two other column "altAllele" and "refAllele". "altAllele" column represents the mutant allele; "refAllele" column represents the reference allele. Required.

Value

Returns a list containing 12 submatrices with different mutation types. ("altAllele" column and "refAllele" column are removed)

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
```

5.3 correct: mitochondrial genotype matrix modification

Description

Using mitochondrial variants frequency matrix as input, transform the zeroes into ones when the median of the non-zero frequencies in the same row ≥0.6.

Usage

```
correct(x)
```

Arguments

x	A dataframe containing mitochondrial variants frequency matrix where a column represented a single cell and a row represented variants frequency of a specific mitochondrial genotype. Required.

Value

Returns a dataframe containing a modified mitochondrial genotype matrix.

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
d2=data.frame()
for(i in 1:12){
    di=d[[i]]
    di2=as.data.frame(t(apply(di,1,correct)))
    d2=rbind(d2,di2)
}
```

5.4 top_sites: highly variable sites identification

Description

Using mitochondrial variants frequency matrix as input, the 20/50 highest variable sites were called by identifying the rows with highest standard

deviation (sd) across columns(cells).

Usage

```
top_sites(d)
```

Arguments

d	A dataframe containing mitochondrial variants frequency matrix where a column represented a single cell and a row represented variants frequency of a specific mitochondrial genotype. Required.

Value

Returns a dataframe containing 2 modified mitochondrial genotype matrix with 20/50 highest variable sites.

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
tops=data.frame()
tops2=data.frame()
for(i in 1:12){
    di=d[[i]]
    tops=rbind(tops,top_sites(di)[[1]])
    tops2=rbind(tops2,top_sites(di)[[2]])
}
```

5.5 learn_distance: distance calculation in subspaces

Description

By integrating distance information and consensus information, *learn_distance* generate a more integrative distance matrix**.**

Usage

```
learn_distance(
    c,
    pclusters,
    diss
    )
```

Arguments

c	Integer. 1:(number of selected subspaces). Required.
pclusters	A list containing clustering results in selected subspaces with kmeans. Required.
diss	A list containing Euclidean distance matrixes of selected subspaces. Required.

Value

Returns a dataframe containing a more integrative distance matrix.

Examples

```
data("pcluster")
```

```
data("diss")
data("cs")
c=1:length(cs)
dis=learn_distance(c,pcluster,diss)
```

5.6 find_markers: marker variants identification

Description

find_markers performs evaluation of variant as marker for the consensus clustering result by calculating the receiver operating characteristics (AUC) score and Pearson's correlation coefficient between the binary cluster labels and the frequency distribution.

Usage

```
find_markers(
      f,
      cluster
      )
```

Arguments

f	A list containing the frequency distribution of a variant. Required.
cluster	A list containing the consensus clustering result. Required.

Value

Returns a dataframe containing AUC score and p-value of the given variant.

Examples

```
data("TF1_clones")
data("scc")
data=TF1_clones$data
d=data_prepare(data)
tops2=data.frame()
for(i in 1:12){
      di=d[[i]]
      tops2=rbind(tops2,top_sites(di)[[2]])
}
find_markers_2=function(x,cluster=scc){
      a=find_markers(as.numeric(as.character(x)),cluster=as.numeric(as.c
haracter(scc)))
      a=a[a$p==min(a$p),]
      return(a)
}
markers=apply(tops2,1,find_markers_2)
```

5.7 sum_score: S_{score} and D_{score} calculation for optimization

Description

Calculate S_{score} and D_{score} for iterative optimization.

Usage

```
sum_score(
    score,
    cs,
    aris
    )
```

Arguments

score	A dataframe containing adjusted rand indexes between the refined clustering results (resulted from t-SNE or UMAP-Kmeans procedures) and the clustering results in the selected subspaces. Required.
cs	Indexes of selected subspaces. Required.
aris	A dataframe containing ARI values among effective subspaces. Required.

Value

Returns a list containing S_{score}, suggested dimensional reduction method, number of effective subspaces and average ARI values used in D_{score} calculation.

Examples

```
data("score")
data("cs")
data("aris")
score_sug=sum_score(score, cs, aris)
```

5.8 main: main function

Description

main is the main part of LINEAGE for lineage analysis.

Usage

```
main(
    d,
    centers = 3,
    nmarker = 16
    )
```

Arguments

d	A list containing 12 submatrices with different mutation types. Output of *data_prepare()*. Required.
centers	Integer. The number of clusters used in Kmeans procedure. Default: 3.
nmarker	Integer. The number of markers showed in final result. Default: 16.

Value

Returns a list containing lineage analysis result.

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
a=dim(d[[1]])[2]
if(a>100){
    nmarker=c(15,20)
    centers=3
}else{
    nmarker=c(10,15)
    centers=2
}
result=main(d, centers, nmarker)
```

5.9 best_iter: non-parallel iterative optimization

Description

best_iter performs a non-parallel iteration process of *main* to guarantee a more stable and reliable cell clustering/clone tracing result.

Usage

```
best_iter(
d = d,
centers = c(2,3),
nmarker = c(10,15,20),
repeats = 30
)
```

Arguments

d	A list containing 12 submatrices with different mutation types. Output of *data_prepare()*. Required.
centers	Integer. The number of clusters used in Kmeans procedure. Default: c(2,3).
nmarker	Integer. The number of markers showed in final result. Default: c(10,15,20).
repeats	Integer. The number of iterations. Default: 30.

Value

Returns a list containing results of all repeats, including the best result.

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
a=dim(d[[1]])[2]
if(a>100){
    nmarker=c(15,20)
    centers=3
```

```
}else{
    nmarker=c(10,15)
    centers=2
}
all_results= all_results=best_iter(d, centers, nmarker, repeats=30)
```

5.10 multi_best_iter: parallel iterative optimization

Description

multi_best_iter performs a parallel iteration process of *main* to guarantee a more stable and reliable cell clustering/clone tracing result.

Usage

```
multi_best_iter(
d = d,
centers = c(2,3),
nmarker = c(10,15,20),
repeats = 30,
thread = 10
)
```

Arguments

d	A list containing 12 submatrices with different mutation types. Output of *data_prepare()*. Required.
centers	Integer. The number of clusters used in Kmeans procedure. Default: c(2,3).
nmarker	Integer. The number of markers showed in final result. Default: c(10,15,20).
repeats	Integer. The number of iterations. Default: 30.
thread	Integer. The number of threads to run *multi_best_iter*. Default: 10.

Value

Returns a list containing results of all repeats, including the best result.

Examples

```
data("TF1_clones")
data=TF1_clones$data
d=data_prepare(data)
a=dim(d[[1]])[2]
if(a>100){
    nmarker=c(15,20)
    centers=3
}else{
    nmarker=c(10,15)
    centers=2
}
all_results=multi_best_iter(d, centers, nmarker, repeats=30, thread=10)
```

5.11 lineage_tree: trace the lineage tree

Description

Using the result of best_iter or multi_best_iter as input, lineage_tree will export lineage tree of the lineage analysis result.

Usage

```
lineage_tree(
    result
    )
```

Arguments

result	The result of best_iter or multi_best_iter. Required.

Value

Returns an object of class hclust which describes the lineage tree produced by LINEAGE.

Examples

```
data("TF1_clones")
data=TF1_clones$data
## performs a non-parallel iteration process
result=lineage(data, repeats=30, thread=NULL)
## performs a parallel iteration process
result=lineage(data, repeats=30, thread=10)
hc=lineage_tree(result)
```

5.12 traceplot: visualization of the result

Description

Using the result of best_iter or multi_best_iter as input, traceplot will export a heatmap and a suggested 2D visualization plot for the lineage analysis result.

Usage

```
traceplot(
    result,
    label
    )
```

Arguments

result	The result of best_iter or multi_best_iter. Required.
label	A dataframe containing lineage label with at least 2 columns named "Sample" and "label". "Sample" column contains sample ids; "Label" column is the lineage label of each sample. Required.

Value

Returns a list containing a heatmap and a 2D visualization plot for the result of LINEAGE.

Examples

```
data("TF1_clones")
data=TF1_clones$data
rlabel=TF1_clones$rlabel
## performs a non-parallel iteration process
result=lineage(data, repeats=30, thread=NULL)
## performs a parallel iteration process
result=lineage(data, repeats=30, thread=10)
## plots with inferred clone labels
plots=traceplot(result = result, label= result$label)
## plots with reference clone labels
plots=traceplot(result = result, label= rlabel)
```

6 Demos

We have provided an example data "TF1_clones.rda" containing a mitochondrial genotype matrix of TF1_clones and its validated cell lineage information in the *data* dictionary of LINEAGE package, which can be used for test. It should be noted that the result has a certain randomness because of the randomness from clustering and dimension reduction processes.

6.1 mode1: run parallel iterative optimization

```
data("TF1_clones")
data=TF1_clones$data
result=lineage(data = data, repeats = 30, thread = 20)

[1] "parallel processing"
[1] "start to choose best parameters for c(centers, nmarker)"
[1] "the best centers is 2"
[1] "the best num of markers is 10"
[1] ">score results of all repeats with different 'centers' and 'num of
markers'"
[1]     "centers     num_of_markers     Sscore(score)     Dscore(score1)
averageAUC(score2) num_of_selected_subspaces"
[1]  2.000000 10.000000  7.214836  4.903350  9.245944  2.000000
……
[1] ">max Sscore of all repeats with different 'centers' and 'num of
markers'"
[1] 7.214836
```

6.2 mode2: run non-parallel iterative optimization

```
data("TF1_clones")
data=TF1_clones$data
result=lineage(data = data, repeats = 30, thread = NULL)

[1] "non-parallel processing"
[1] "start to choose best parameters for c(centers, nmarker)"
[1] "run the 1 th iteration"
[1] "run the 2 th iteration"
[1] "run the 3 th iteration"
……
[1] "the best centers is 2"
[1] "the best num of markers is 10"
[1] ">score results of all repeats with different 'centers' and 'num of
```

```
markers'"
[1]     "centers     num_of_markers     Sscore(score)     Dscore(score1)
averageAUC(score2) num_of_selected_subspaces"
[1]  2.000000 10.000000  9.125904  6.831701  9.176814  4.000000
……
[1] ">max Sscore of all repeats with different 'centers' and 'num of
markers'"
[1] 9.125904
```

6.3 trace the lineage tree

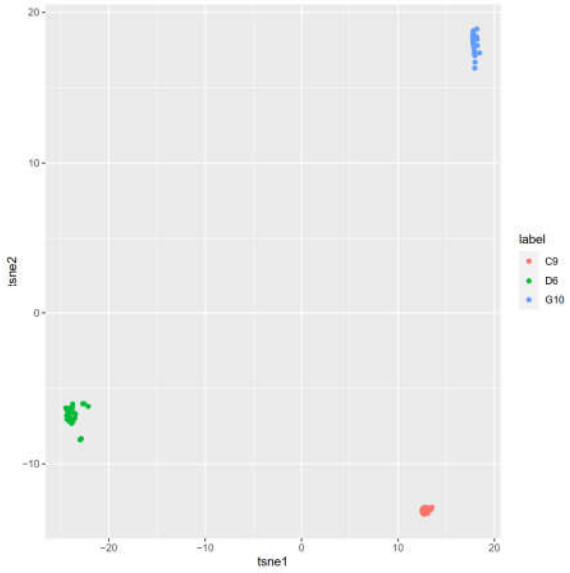
```
data("TF1_clones")
data=TF1_clones$data
result=lineage(data = data, repeats = 30, thread = 20)

# The inferred clone labels are embedded in the returned
# result as result$label. We also provide lineage_tree function to
# trace the lineage tree of the returned result.
hc=lineage_tree(result)
```

```
> str(hc,max.level=1)
List of 7
 $ merge      : int [1:69, 1:2] -48 -64 -47 -46 -56 -59 -58 -13 -49 -55 ...
 $ height     : num [1:69] 5.15e-05 1.62e-04 1.69e-04 2.75e-04 3.55e-04 ...
 $ order      : int [1:70] 42 23 32 31 43 22 45 25 26 35 ...
 $ labels     : chr [1:70] "SRR7245968" "SRR7245969" "SRR7245970" "SRR7245971" ...
 $ method     : chr "complete"
 $ call       : language hclust(d = as.dist(diss))
 $ dist.method: NULL
 - attr(*, "class")= chr "hclust"
```

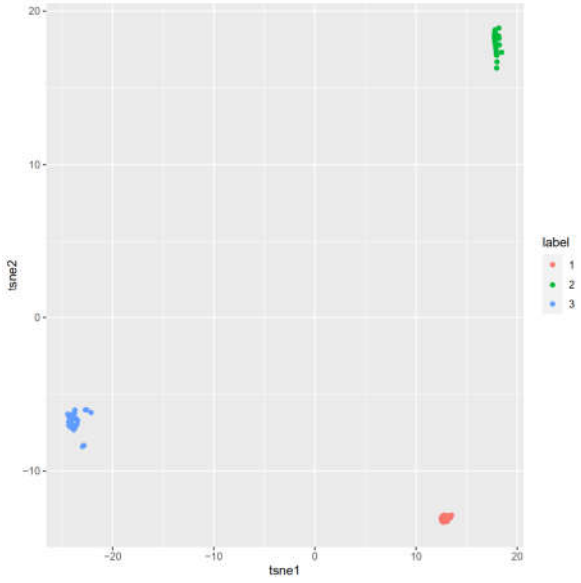
6.4 visualization of the result

```
data("TF1_clones")
data=TF1_clones$data
label= TF1_clones$rlabel    #reference clone labels
result=lineage(data = data, repeats = 30, thread = 20)

plots=traceplot(result, label)  #plots with reference clone labels
plots=traceplot(result, result$label)  #plots with inferred clone labels

# 2d visualization cluster results with clonal labels
# colored in reference clone labels or inferred labels
print(plots$d2d)
# Heatmap results with markers information across cells and color bar is
# colored in reference clone labels
print(plots$heatmap)
```

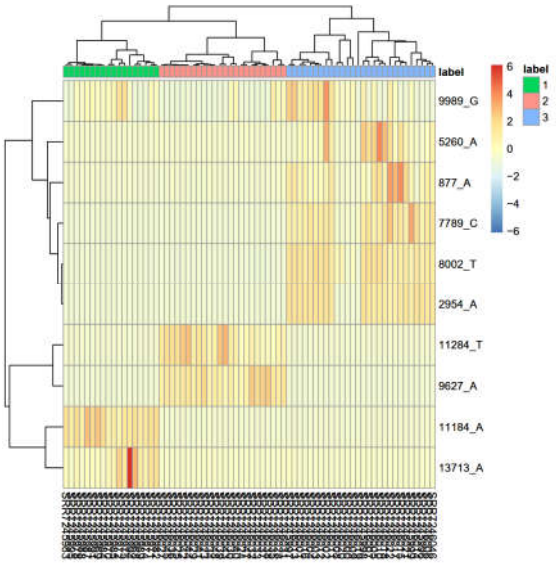
√ 2d.pdf (colored in reference clone labels)



√ 2d.pdf (colored in inferred labels)



√ heatmap.pdf (colored in inferred labels)



6.5 results

√ result

```
# The recommended result is embedded in the returned result as result$best.
> best=list(result=result$best, plots=plots)
> str(best, max.level=1)
List of 2
 $ result:List of 14
 $ plots :List of 2
```

```
> str(result$best,max.level=1)
List of 14
 $ tsnes    :'data.frame':      70 obs. of  2 variables:
 $ umaps    :'data.frame':      70 obs. of  2 variables:
 $ markers  :'data.frame':      10 obs. of  70 variables:
 $ use      : Named logi [1:70] TRUE TRUE TRUE TRUE TRUE TRUE ...
  ..- attr(*, "names")= chr [1:70] "SRR7245968" "SRR7245969" "SRR7245970" "SRR7245971" ...
 $ suggest  : chr "tsne"
 $ score    : num 8.31
 $ fig1     :List of 9
  ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
 $ fig2     :List of 9
  ..- attr(*, "class")= chr [1:2] "gg" "ggplot"
 $ centers  : num 2
 $ nmarker  : num 10
 $ score1   : num 5.97
 $ score2   : num 9.34
 $ nsubspace: int 4
 $ saris    : num 0.379
```

7 Suggestions

lineage: Considering the randomness from clustering and dimension reduction processes, an iteration process with at least 30 repeats(default) is recommended to guarantee a more stable and reliable cell clustering/clone tracing result.