scDDboost

Xiuyu Ma

Contents

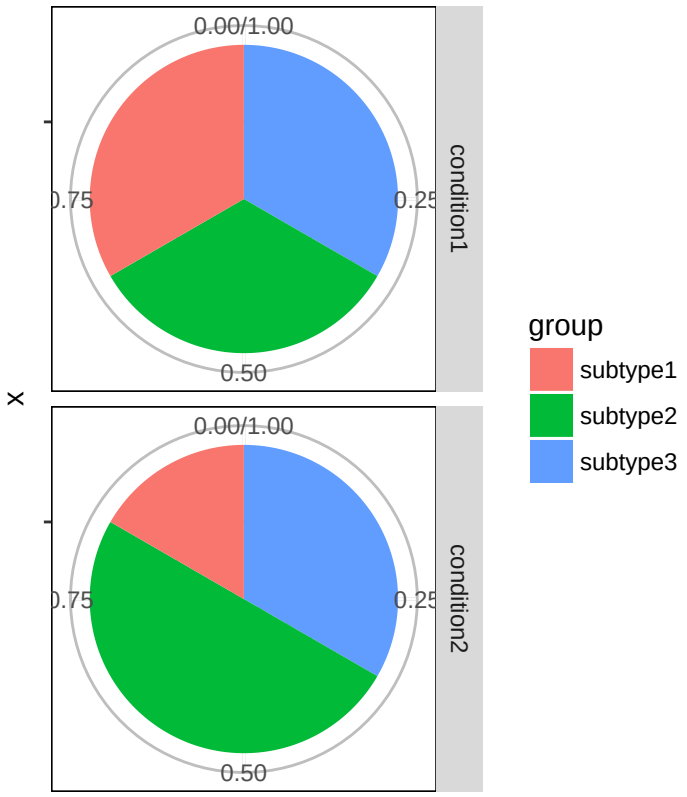
Abstract:

The scDDboost package models single-cell gene expression data (from single-cell RNA-seq) using mixture models in order to explicitly handle heterogeneity within cell populations. In bulk RNA-seq data, where each measurement is an average over thousands of cells, distributions of expression over samples are most often unimodal. In singlecell RNA-seq data, however, even when cells represent genetically homogeneous populations, multimodal distributions of gene expression values over samples are common [1]. This type of heterogeneity is often treated as a nuisance factor in studies of differential expression in single-cell RNA-seq experiments. Here, we explicitly accommodate it in order to improve power to detect differences in expression distributions that are more complicated than a mean shift.

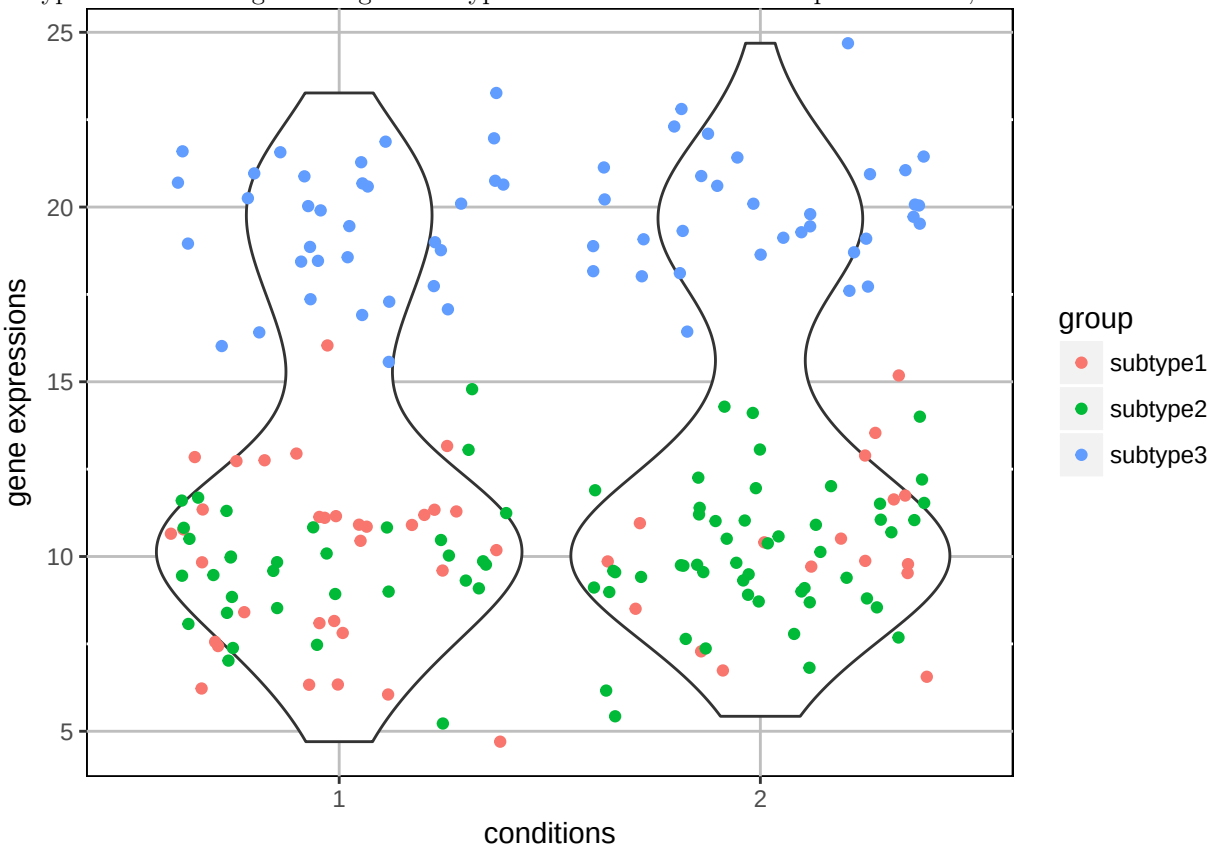
Package scDDboost 1.0.0 Report issues on "github link here"

Background 1

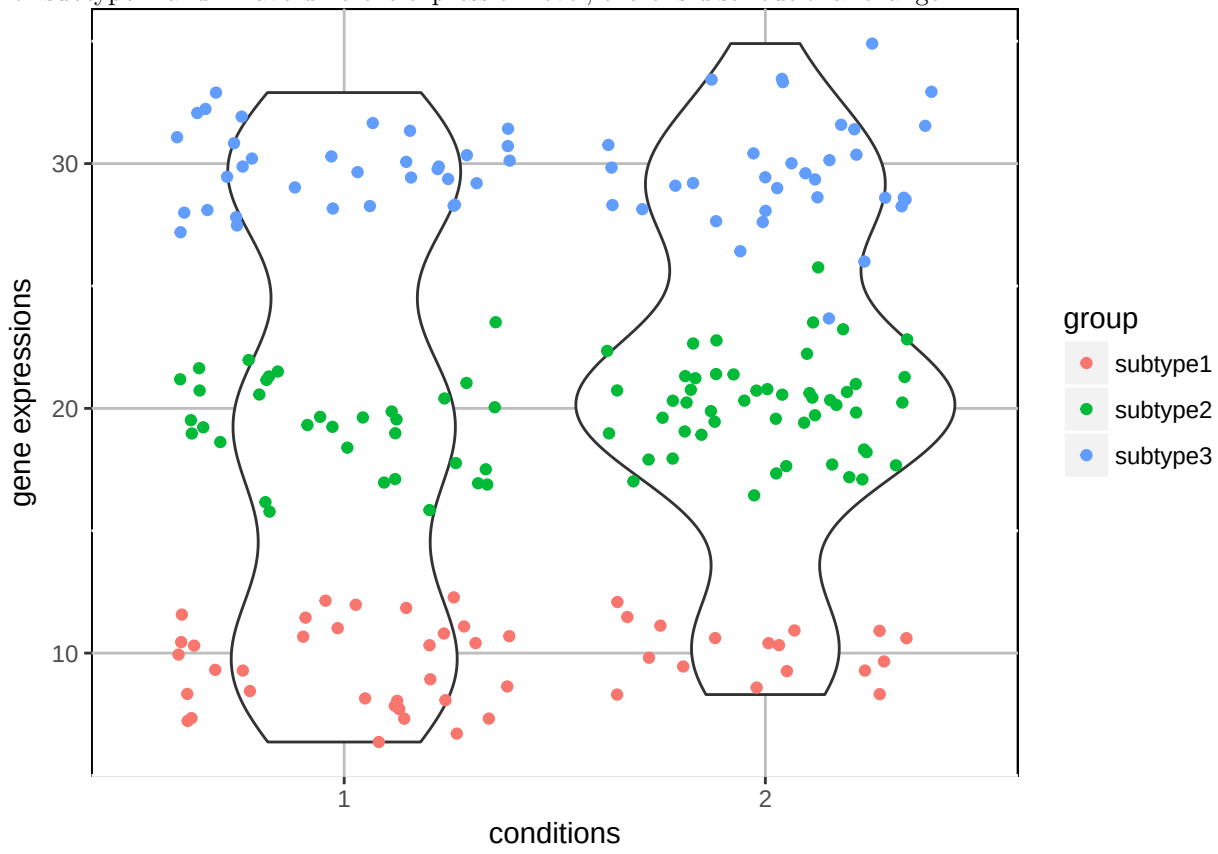
Our aim is to identify differential distributed genes across two biological conditions. We view samples as mixture of cells coming from distinct subtypes. Change of proportions of subtypes does not necessarily lead to distributional change of transcripts across conditions, given that subtypes could share lots of genes with equivalent expressed and for two subtypes as long as their total proportion remains unchanged across biological conditions, the change of individual proportions would not lead distributional change at those



equivalent expressed genes. even proportions of subtypes 1 and 2 changed. For gene subtype 1 and 2 have the same expression level, there is no DD.



For subtype 1 and 2 have different expression level, there is distributional change



Identify DD genes 2

In this section, we demonstrate how to cluster cells by modal cluster and how to use the main function **PDD** to find genes with differential distributions \ First we need to load the scDDboost package. For each of the following sections in this vignette, we assume this step has been carried out.\

```
library(scDDboost)
```

```
## Loading required package: EBSeq
```

```
## Loading required package: blockmodeling
```

```
## Loading required package: gplots
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
## Loading required package: testthat
```

```
## Loading required package: Oscope
```

```
## Loading required package: cluster
```

```
## Loading required package: BiocParallel
```

```
## Loading required package: mclust
```

```
## Package 'mclust' version 5.3

## Type 'citation("mclust")' for citing this R package in publications.

## Loading required package: doParallel

## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

Next, we load the toy simulated example a *SingleCellExperiment* object that we will use for identifying and classifying DD genes.\

```
data(sim_dat)
```

Verify that this object is a member of the SingleCellExperiment class and that it contains 200 samples and 500 genes. The colData slot (which contains a dataframe of metadata for the cells) should have a column that contains the biological condition or grouping of interest. In this example data, that variable is the 'condition' variable. Note that the input gene set needs to be in SingleCellExperiment format, and should contain normalized counts. In practice, it is also advisable to filter the input gene set to remove genes that have an extremely high proportion of zeroes. First we identify subtypes of cells, to do so we compute distance matrix between cells.

```
library(SummarizedExperiment)
```

```
## Loading required package: GenomicRanges

## Loading required package: stats4

## Loading required package: BiocGenerics

##
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, cbind, colMeans,
##     colnames, colSums, do.call, duplicated, eval, evalq, Filter,
##     Find, get, grep, grepl, intersect, is.unsorted, lapply,
##     lengths, Map, mapply, match, mget, order, paste, pmax,
##     pmax.int, pmin, pmin.int, Position, rank, rbind, Reduce,
##     rowMeans, rownames, rowSums, sapply, setdiff, sort, table,
##     tapply, union, unique, unsplit, which, which.max, which.min

## Loading required package: S4Vectors

##
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:gplots':
##
```

```
##     space
## The following object is masked from 'package:base':
##
##     expand.grid
## Loading required package: IRanges
## Loading required package: GenomeInfoDb
## Loading required package: Biobase
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
## Loading required package: DelayedArray
## Loading required package: matrixStats
##
## Attaching package: 'matrixStats'
## The following objects are masked from 'package:Biobase':
##
##     anyMissing, rowMedians
##
## Attaching package: 'DelayedArray'
## The following objects are masked from 'package:matrixStats':
##
##     colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges
## The following object is masked from 'package:base':
##
##     apply
```

```
data_count = assays(sim_dat)$counts
```

```
## Loading required package: SingleCellExperiment
```

```
conditions = colData(sim_dat)$conditions
rownames(data_count) = 1:500
##here we use 1 core to compute the distance matrix
D = cal_D(data_count,1)
##number of subtypes
K = 4
ccl = pam(D,k = 4)$clustering
```

After we identify subtypes of cells, we use EBSeq to find the probabitlities of each DE pattern among subtypes.

```
##hyper parameters for EBSeq
hp = rep(1, 1 + nrow(data_count))

##partition patterns
Posp = pat(K)[[1]]

##cluster of genes, if not just set each gene to form one cluster.
gcl = 1:nrow(data_count)
```

```
##size factor, for normalized counts set all size factor to be 1

sz = rep(1, ncol(data_count))

##posterior probability of diffferential distributed
res = PDD(data_count,conditions, 1, K, D, hp, Posp, iter = 0, random = F, nrandom = 0)
```

If we set threshold to be 5% then we have estimated DD genes

```
EDD = which(res > 0.95)
```

Determine number of clusters