Rank Based Differential Expression Analysis (RBDA)

Ryne Ramaker

2019-10-03

Overview

The primary function of this package is identify differentially expressed genes from raw RNA-sequencing count tables. The primary function *RBDA* uses a non-parametric, rank based approach to finding differentially expressed genes between sample groups or between a single sample and a large reference database. This function is optomized to run efficiently on large sample sizes of >1000 samples.

For sample group comparisons, the variation in raw count rank within a sample group, V_{tr}, can be described as:

$$V_{tr} = \sum_{i=1}^{n} (x_{it} - \mu_{tr})^2$$

Where n is the number of samples in the group, x_{it} is the rank in terms number of reads/fragments assigned to the ith replicate for the tth gene/tanscript, and μ_{tr} is the mean rank of the tth gene/transcript in the rth sample group. Rank ties (usually occuring at low count ranks) are assigned a random order.

Similarly, total variation in count rank for each gene/transcript (t) across sample groups, D_t, can be described as:

$$D_t = \sum_{i=1}^{n} (x_{it} - \mu_t)^2$$

And a test statistic, S_t, for differentially expressed transcripts across any number of sample groups, a, can be described as:

$$S_t = \frac{D_t}{\sum_{r=1}^{a} V_{tr}}$$

Probabilities of observing a given test statistic by chance alone (P-values) are computed by randomly shuffling the count table and computing a large number of null test statistics.

In comparisons of a single sample to a large reference set, computing S_t can be simplified to the following: $S_t = \frac{D_t}{V_{tR}}$

where R represents the large reference set not included the single sample designated for comparison.

Installation

There are multiple ways to install this package. The first involves downloading the binary source file from my github page (https://github.com/rramaker/RBDA) and installing the package manually:

```
install.packages("/PathToDownloadedFile/RBDA-master.zip")
```

The package can be directly installed from github with the devtools package

```r
library(devtools)
install_github("rramaker/RBDA")
```

Alternatively the package can be downloaded from the CRAN repository. ##Coming Soon!

```r
install.packages("RBDA")
```

Once installed the functions from RBDA can be made available in your current working environment with the library() command:

```r
library(RBDA)
```

Formatting data

RNA-sequencing count table

The primary input to the *RBDA* function is a dataframe or matrix containing raw count data with samples as columns and genes/transcripts as rows. Gene/transcript names can be indicated as row names. Sample IDs can be indicated as column names. An example raw count input file is provided with the package and shown below:

```r
data("BRCA_Counts")

BRCA_Counts[1:5,1:3]
```

```
##                V1  V2  V3
## ?|100130426     0   0   0
## ?|10357       353 362 193
## ?|136542        0   0   0
## ?|340602        2   0   0
## ?|390284       21  17  40
```

Sample information table

The second required input to the *RBDA* function is a dataframe containing information about which group labels for the samples contained in the count table. The rows of the this table should represent each sample in the count table and the columns should be factor variable indicating to which group each sample belongs. The row names of this table must correspond to the column names of the raw count table above. An example generating the sample information table for the count table above is shown below:

```r
colData<-data.frame(Type=as.factor(c(rep(0,5),rep(1,5))),row.names=colnames(BRCA_Counts))

colData
```

```
##     Type
## V1     0
## V2     0
## V3     0
## V4     0
## V5     0
## V6      1
## V7      1
## V8      1
## V9      1
## V10     1
```

Performing differential expression analysis

Finding differential expression between sample groups

Differential expression analysis based on the sample variable specified in the sample information file can be performed using the *RBDA* function as shown below:

```
Results<- RBDA(countData = BRCA_Counts, colData=colData, testVariable = "Type")

head(Results)
```

```
##              variance_rank test_statistic  p_value
## ?|100130426          25.04       1.179855 0.262860
## ?|10357              23.32       1.062779 0.507235
## ?|136542             10.60       1.079175 0.456375
## ?|340602             58.16       1.051891 0.546835
## ?|390284             11.86       1.024574 0.679090
## ?|391343             84.02       1.112476 0.375840
```

The resulting output is a dataframe containing genes/transcripts as rows with three columns. The first column is the percentile rank of variance of each sample in the largest sample group. Low numbers indicate samples with very low sample variance. The second column is the test_statistic, S_t, described above. The third column is the p-value generated from random shuffling of the count table columns as described above.

Finding differential expression between a single sample and reference group

Comparisons of a single sample to a reference set can be performed similarly to the group-based analysis above with the exception that the reference set of samples must be assigned the first factor level in the sample information table. We recommend creating a simple numbered factor variable (0,1) to avoid confusion. In the example below the 10th sample in the test dataset "BRCA_Counts" is treated as the sample of interest and samples 1-9 are treated as the reference group.

```
colData<-data.frame(Type=as.factor(c(rep(0,9),rep(1,1))),row.names=colnames(BRCA_Counts))

Results<- RBDA(countData = BRCA_Counts, colData=colData, testVariable = "Type")

head(Results)
```

```
##              variance_rank test_statistic  p_value
## ?|100130426          22.82       1.014802 0.733475
## ?|10357              18.06       1.592938 0.064375
## ?|136542             34.80       1.026924 0.643805
## ?|340602             67.84       1.087800 0.406030
## ?|390284              5.70       1.011107 0.767550
## ?|391343             75.42       1.024174 0.661815
```

We envision the best apporach for N=1 experiments will include a batch of co-sequenced, potentially related unaffected samples in addition to the sample of interest to compare to the larger reference set of samples. These samples can be used to filter differences that are not specific to the sample of interest, rather are related to the family or batch. This type of analysis be performed by creating a second column in the sample information table specifying samples with family or batch relation to the sample of interest. Reference data set must be the first factor level in this columns as well. The example below is identical to the previous except for the second column in the sample information table indicating the samples 8-10 were sequenced as a batch:

```
colData<-data.frame(Type=as.factor(c(rep(0,9),rep(1,1))),Batch = as.factor(c(rep (0,7), rep(1,3))),row.

Results<- RBDA(countData = BRCA_Counts, colData=colData, testVariable = "Type",batch_family_variable =

head(Results)
```

```
##              variance_rank test_statistic  p_value min_batch_fam_p_value
## ?|100130426         22.82       1.014802 0.733475              0.166545
## ?|10357             18.06       1.592938 0.064375              0.264925
## ?|136542            34.80       1.026924 0.643805              0.041515
## ?|340602            67.84       1.087800 0.406030              0.052600
## ?|390284             5.70       1.011107 0.767550              0.132900
## ?|391343            75.42       1.024174 0.661815              0.650120
```

As shown above this results in a fourth output column which indicates the minimum p-value obtained by hypothetically using each of the specified batch samples were used as the sample of interest allowing the user to perform downstreaming filtering accordingly.

Improving performance with parallelization

The *RBDA* function makes use of parallelization with R's parallel package. The number of cores used by the function can be changed with the *numCores* flag. We have found most computers can readily handle 4 cores (the default), but this can be increased on systems with higher capacity. The number of cores available on a computer can be determined as shown below:

```
library(parallel)
detectCores()
```

```
## [1] 4
```