Reference manual: RaceID (Rare Cell Type Identification)

0. Prerequisites

This manual describes in detail how to run the R-code for the RaceID (**Ra**re **Ce**ll Type **ID**entification) algorithm, including k-means clustering of single cell expression data, identification of outlier cells, and inference of the final clusters that represent distinct cell types or cell states.

The algorithm is implemented as an `S4` class object, named `SCseq`. The R script `RaceID_class.R` contains the specifications of the class object and all methods. The additional R script `RaceID_sample.R` lists sample commands to run a complete analysis on example data. These data are provided as text file `transcript_counts_intestine.xls` and should be stored in the working directory together with the script `RaceID_class.R`. After opening an R workspace in the same directory the commands in `RaceID_sample.R` can be executed.

The algorithm requires a number of R packages that have to be installed:

```
>install.packages(c("tsne", "pheatmap", "MASS", "cluster", "mclust",
"flexmix", "lattice", "fpc",  "RColorBrewer", "permute", "amap", “locfit”))
```

Prior to running the code these packages have to be loaded into the workspace:

```
> require(tsne)
> require(pheatmap)
> require(MASS)
> require(cluster)
> require(mclust)
> require(flexmix)
> require(lattice)
> require(fpc)
> require(amap)
> require(RColorBrewer)
> require(locfit)
```

The `S4` class object is specified in the R script `RaceID_class.R`, which has to be loaded into the workspace. If the file is deposited in the working directory, this is done by the command:

```
> source("RaceID_class.R")
```

Otherwise, the full path has to be indicated.

Please note the final paragraph “Remarks on running RaceID” containing important advice on how to choose parameters for running RaceID.

I. Input data

Input data have to be provided as a `data.frame` object with expression values, i. e. raw read or transcript counts, with genes as rows and cells as columns. Row names should correspond to gene identifiers and column names to identifiers of single cell samples. It is recommended to remove any spike-in RNAs from the expression table, if clustering due to variability of non-endogenous transcripts should be avoided.

The provided example data can be loaded using the following commands:

```
> x <-read.csv("transcript_counts_intestine.xls", sep="\t", header=TRUE)
```

The first column contains unique gene ids and has to assigned as rownames:

```
> rownames(x) <- x$GENEID
```

The gene ids of the non-endogenous spike-in RNAs starts with “ERCC” and these transcripts have to be removed:

```
> prdata <- x[grep("ERCC",rownames(x),invert=TRUE),-1]
```

Now, the input data have the correct format:

```
> head(prdata[,1:4])
                          I_1      I_2      I_3      I_4
0610005C13Rik__chr7  2.007853 1.001958 0.000000 5.049473
0610007N19Rik__chr15 0.000000 0.000000 0.000000 0.000000
0610007P14Rik__chr12 1.001958 0.000000 1.001958 3.017717
0610008F07Rik__chr2  0.000000 0.000000 0.000000 1.001958
0610009B14Rik__chr12 0.000000 0.000000 0.000000 0.000000
0610009B22Rik__chr11 1.001958 0.000000 1.001958 0.000000
```

The primary input data object is denoted as `prdata` in the following.

Given a data.frame `prdata` of transcript counts, an `SCseq` object can be created:

```
> sc <- SCseq(prdata)
```

The transcript count data.frame initializes the slots `sc@expdata`, `sc@ndata`, and `sc@fdata`. The contents of a slot, for example `expdata`, can be written out by the command:

```
> sc@expdata
```

II. Preprocessing of input data

As a first step the input data are subject to filtering by applying the method `filterdata`.

```
> sc <- filterdata(sc, mintotal=3000, minexpr=5, minnumber=1, maxexpr=500,
downsample=FALSE, dsn=1, rseed=17000)
```

The function call displays the default values of all arguments.
After discarding cells with less than `mintotal` transcripts, the expression data are normalized. Normalization is performed by dividing transcript counts in each cell by the total number of transcripts in this cell followed by multiplication with the median of the total number of transcripts across cells. If `downsample` is set to `TRUE`, than transcript counts are downsampled to `mintotal` transcripts per cell, instead of the normalization. Downsampled versions of the transcript count data are averaged across `dsn` samples. If `dsn` equals one, sampling noise should be comparable across cells, while for high numbers of `dsn` the data will become similar to the median normalization. To ensure reproducibility of downsampling the random number generator is initialized with a fixed seed `rseed` that can be changed.
After normalization or downsampling a pseudocount of `0.1` is added to the expression data, and the output overwrites the slot `sc@ndata`.
Finally, genes that are not expressed at `minexpr` transcripts in at least `minnumber` cells are discarded.
If expression data are derived based on unique molecular identifiers[1–3] (UMIs) it is recommended to filter out genes with saturated UMI counts, since differences in saturation between cells will affect the transcriptome correlation and potentially introduce spurious clusters. This is done by discarding genes with at least `maxexpr` transcripts in at least a single cell after normalization or downsampling. To disable this filtering step, `maxexpr` should be set to `Inf`. After gene filtering, the transcript count data overwrite the slot `sc@fdata`.
All filtering parameters used are written to the slot `sc@filterpar`.

III. k-means clustering

The preprocessed data are clustered by k-means clustering.

```
> sc <- clustexp(sc, metric="pearson", cln=0, do.gap=TRUE, clustnr=20,
B.gap=50, SE.method="Tibs2001SEmax", SE.factor=.25, bootnr=50, rseed=17000)
```

The function call displays the default values of all arguments. These arguments define how the clustering is performed when calling the method:

`metric:`	the input data `sc@fdata` are transformed to a distance object. Distances can be computed based on different metrics. Possible values are `"pearson"`, `"spearman"`, `"kendall"`, `"euclidean"`, `"maximum"`, `"manhattan"`, `"canberra"`, `"binary"` or `"minkowski"`. Default is `"pearson"`. In case of the correlation based methods, the distance is computed as 1 – correlation. The distance object is written to the slot `sc@distances`. K-means clustering is performed on this distance object.
`cln:`	the number of clusters for k-means clustering. Default is `0`. In this case, the cluster number is determined based on the gap statistics[4] and `do.gap` has to be `TRUE`.
`do.gap:`	if `do.gap` equals `TRUE` the number of clusters is determined based on the gap statistics[4], which is defined as the difference between within cluster dispersion of the original data and a background distribution modeled by a uniform distribution within the limits of the original data. Default is `TRUE`.
`clustnr:`	maximum number of clusters for the computation of the gap statistics. Default is `20`. If more major cell types are expected a higher number should be chosen.
`B.gap:`	number of bootstrap runs for the calculation of the gap statistics. Default is `50`.
`SE.method:`	the clustering routine calls a modified version of the `maxSE` function from the `cluster` package to determine the first local maximum of the gap statistics. By default, we use the method `"Tibs2001SEmax"` for calling the first local maximum (see specification of `maxSE`). This method requires that the maximum exceeds the values of its neighbors by a fraction of their standard deviation. This fraction is defined by the parameter `SE.factor`. All methods defined for the original `maxSE` function can also be used.
`SE.factor:`	fraction of the standard deviation by which the local maximum is required to differ from the neighboring points it is compared to. Default is `0.25`.

Given the desired number of clusters, k-means clustering is performed using the `clusterboot` function from the `fpc` package. We created a new interface function `KmeansCBI` that allows for clustering with various metrics. The `clusterboot` function performs bootstrapping of k-means clustering and quantifies the robustness of all clusters by Jaccards similarity.

`bootnr:`	number of booststrapping runs for `clusterboot`. Default is `50`.
`rseed:`	to obtain exactly reproducible results between different runs the random number generator for the `clusterboot` is seeded manually. Default is `17000`.

All filtering parameters used are written to the slot `sc@clusterpar`. Results of the `clustexp` methods are written to the slot `sc@kmeans`, which contains a list of three objects:

`gap`:	a `clusGap` object with gap statistics (`NULL` if `do.gap` is `FALSE`) with default parameters.

`jaccard`:	Jaccard's similarity for each cluster computed by bootstrapping (see below).
`kpart`:	clustering partition computed by k-means clustering.

If `clustexp` was executed with `do.gap=TRUE`, it is recommended to examine the gap statistics:

```
> plotgap(sc)
```

In Figure 1 an example is shown. Here, the algorithm determines a cluster number of six. If the first local maximum is less pronounced it might be advisable to change the cluster number manually. This can be done by setting the argument `cln` to the desired number. A good choice will be the minimal cluster number at which the gap statistics begin to saturate. We note that the outlier calling procedure can introduce new clusters and therefore corrects for an initial underestimation. However, if the initial number of clusters is too low, the clusters will be very heterogeneous and the outlier calling procedure potentially does not perform well.

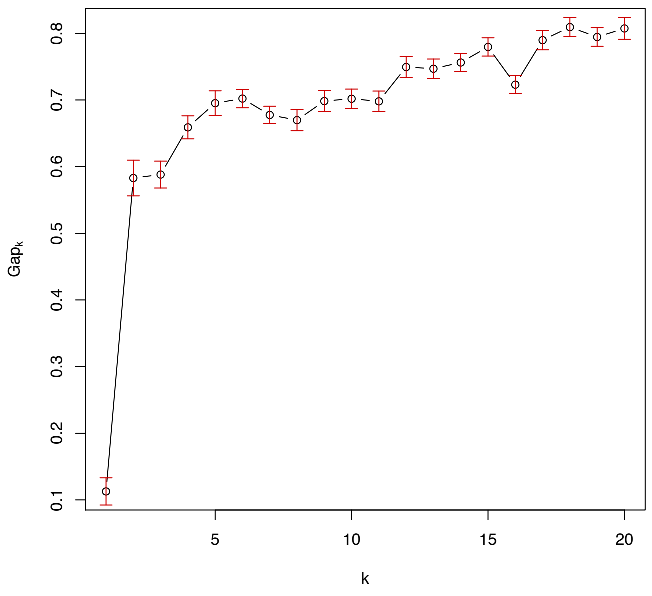

Figure 1. Gap statistics.

Robustness of the clusters can be assessed in various ways. The reproducibility of individual clusters across bootstrapping runs is reflected by Jaccard's similarity (intersect of two clusters divided by the union). A barchart plot of Jaccard's similraties can be generated (see Figure 2):

```
> plotjaccard(sc)
```

Stable clusters should have a Jaccard's similarity greater than 0.6. Clusters with lower values can be resolved by the outlier identification procedure. If too many clusters (more than two) have low values, a smaller number of clusters should be considered by setting the argument `cln` to the desired number.

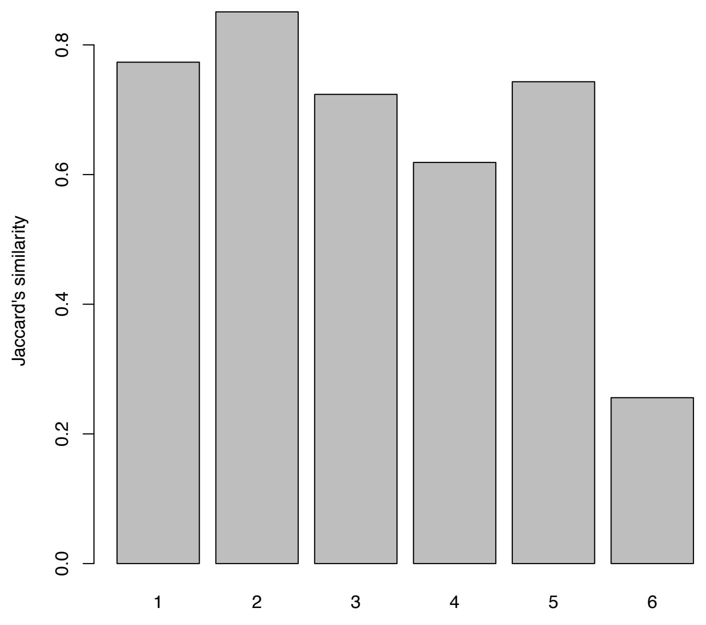

Figure 2. Jaccard’s similarities of k-means clusters.

An alternative method that reflects the resolution of the clustering is a silhouette plot. The silhouette provides a representation of how well each point is represented by its cluster in comparison to the closest neighboring cluster. It computes for each point the difference between the average similarity to all points in the same cluster and to all points in the closest neighboring cluster. This difference it normalize such that it can take values between -1 and 1 with higher values reflecting better representation of a point by its cluster.

A silhouette plot for the k-means clusters can be generated (see Figure 3):

```
> plotsilhouette(sc)
```

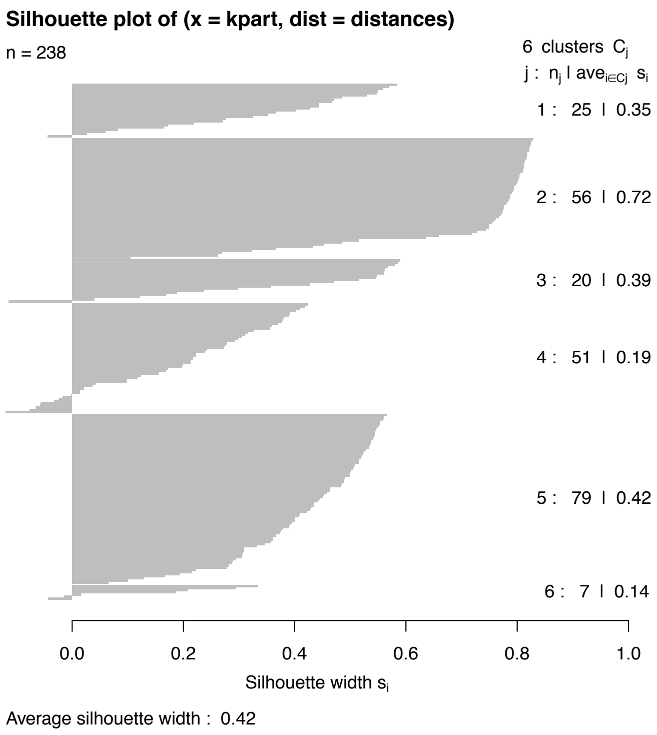

Figure 3. Silhouette plot.

The clusters can also be plotted in a heatmap representation of the cell-to-cell distances (Figure 4):

```
> x <- clustheatmap(sc,final=FALSE,hmethod="single")
```

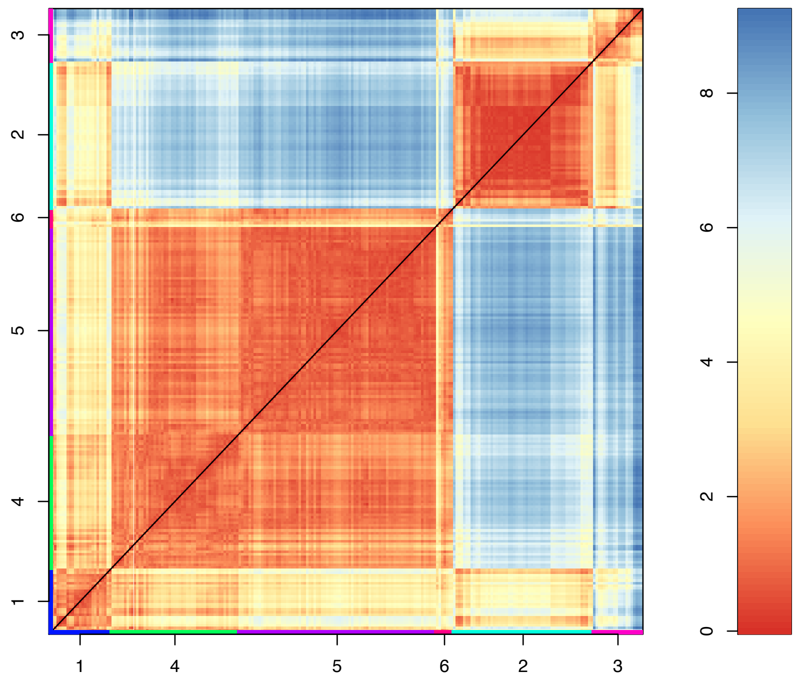

Figure 4. Heatmap of the k-means clusters. Individual clusters are highlighted with rainbow colors along the x- and y-axes.

The logical argument `final` controls if the k-means clusters or the final clusters including the outliers (see below) are plotted. The hierarchical clustering method used for ordering cluster centers can be selected by setting `hmethod` to one of the methods used by the `hclust` function. Default is `"single"`. The output argument is a vector containing the clustering partition in the same order as indicated along the axes of the heatmap. The colours used for highlighting the clusters in this plot and each of the following plots where clusters are highlighted are internally generated in the `clustexp` and the `findoutliers` method, respectively, and stored in the slot `sc@fcol`. If another color scheme is desired a vector with the corresponding color names has to be assigned to the slot. For example:

```
> sc@fcol <- rainbow(6)
```

IV. Outlier identification

After the `clustexp` method has been executed, the outlier identification and the redefinition of the final clusters can be performed by calling the method `findoutliers`:

```
> sc <- findoutliers(sc, outminc=5,outlg=2,probthr=1e-3,thr=2**-(1:40),
outdistquant=.75)
```

The function call displays the default values of all arguments.

`outminc :`	expression cutoff for the identification of outlier genes is defined. Default is `5`.
`probthr:`	defines the probability threshold for outlier calling. If the probability of observing a given expression level for a gene in a cell is lower than this cutoff (based on the negative binomial distribution for the calibrated noise model), the cell is considered an outlier for this gene. Default is 10^{-3}.
`outlg:`	minimal number of outlier genes required to identify a cell as an outlier. Default is `2`.
`outdistquant`:	outlier cells are merged to outlier clusters if their similarity exceeds the `outdistquant`-quantile of the similarity distribution for all pairs of cells that are together in one of the original clusters. Default is `0.75`.
`thr`:	probability values for which the number of outliers is computed in order to plot the dependence of the number of outliers on the probability threshold (see Figure 6).

The method consists of three steps. The first step is the calibration of background model. The outlier identification is based on the distribution of transcript counts within a cluster. For each gene, the transcript count distribution is assumed to be a negative binomial, governed by two parameters: the mean and the dispersion parameter. While the mean is determined directly by averaging expression of a gene across all cells in a cluster, the dispersion parameter is derived from the ensemble of all cells based on the average variance-mean dependence. This dependence is modeled by a second order polynomial in logarithmic space. The approach relies on the idea that the majority of genes are expressed at similar levels across different clusters and biologically meaningful expression differences of a gene between cells in the same cluster should strongly exceed this expected variability.
The regression of the variance-mean dependence and the derivation of the dispersion parameter as a function of the mean is performed in the noise model calibration step of the algorithm. The results are written to the slot `sc@background`, which is a list of three objects:

`vfit`:	object of the class `lm` containing the regression of the variance on the mean.
`lvar:`	function that describes the dependence of the variance on the mean expresssion.

`lsize:`	function that describes the dependence of the dispersion parameter on the mean expresssion.

The quality of the model fit can be inspected by plotting `lvar` as a function of the mean (Figure 5):

```
> plotbackground(sc)
```

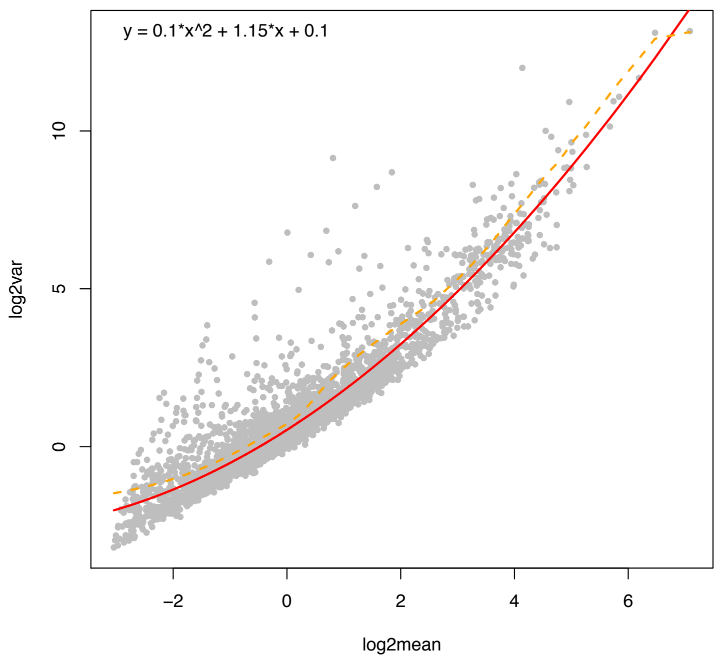

Figure 5. Regression of the variance on the mean by a second order polynomial in logarithmic space.

The plot also contains a local regression for comparison (orange broken line). The polynomial model fit is indicated in the upper left corner.

The second step is the identification of outlier cells. In this step for each gene a probability is computed for the observed transcript counts in every cell of a cluster, inferring the background distribution based on the mean expression in this cluster. If a minimum number of `outlg` genes in a single cell have a transcript count with a probability less than `probthr` after multiple testing correction by the Benjamini-Hochberg method (across cells), this cell is flagged as an outlier. Only genes with more than `outminc` transcripts in at least a single cell within the cluster are included in this test. The results of the outlier detection are written to the slot `sc@out`, which is a list of four elements:

`out:`	a vector with the names of all outlier cells.
`stest:`	a vector with the number of outliers as a function of decreasing values for probthr stored in `thr`.
`thr:`	a vector of probability thresholds used to calculate the number of outliers stored in `stest`.
`cprobs:`	a vector with the probability of the outlier gene with the largest probability, i. e. from a list of genes ranked by outlier probability in increasing order the probability of the gene at rank `outlg` is given.

The number of outliers as a function of the probability threshold can be plotted (Figure 6).

```
> plotsensitivity(sc)
```

The probability threshold should be chosen such that the tail of the distribution is captured as outliers to ensure maximum sensitivity of the method.

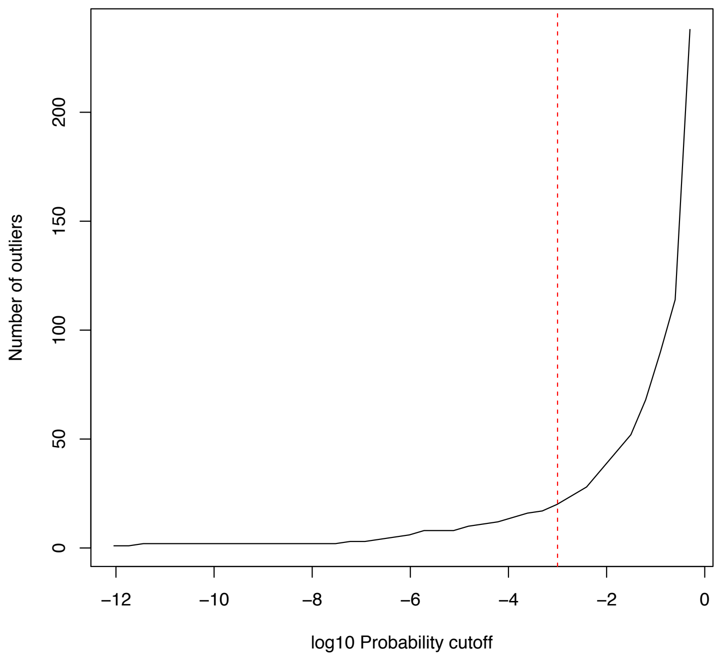

Figure 6. Number of outliers as a function of the probability threshold.

Depending on the expectation for a given dataset the parameter `outlg` can be varied. For instance, if the sensitivity of the sequencing experiment was low and only a few highly expressed genes were reliably quantified it could be necessary to reduce `outlg` to 1, i. e., to require only a single outlier gene to identify a cell as an outlier.

The method `plotoutlierprobs` can be executed to produce a barplot of the outlier probabilities of all cells across cluster (Figure 7):

```
> plotoutlierprobs(sc)
```

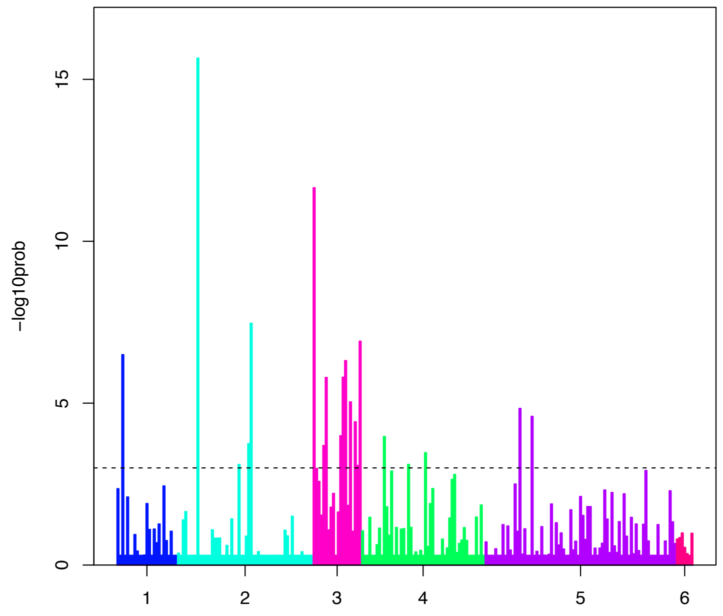

Figure 7. Outlier probabilities of all cells across clusters.

In the final step of the algorithm outlier cells are merged into clusters based on their similarity: outlier cells are merged to outlier clusters if their similarity exceeds the `outdistquant` quantile of the similarity distribution for all pairs of cells within one of the original clusters. After the outlier cells are merged, new cluster centers are defined for the original clusters after removing the outliers and for the additional outlier clusters. Subsequently, each cell is assigned to the nearest cluster center. The final clusters are denoted by increasing number, where the number of the remaining original clusters are the same as in the original partition `sc@kmeans$kpart`. The final partitioning of the cells into clusters is stored in the slot `sc@cpart` and can be written to a text file:

```
> x <- data.frame( CELLID=names(sc@cpart), cluster=sc@cpart )
> write.table(x[order(x$cluster,decreasing=FALSE),], "cell_clust.xls",
row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
```

All parameters used by the method `findoutliers` are stored in the slot `sc@outlierpar`.

V. Analysis of the final clusters

The function `clustdiffgenes` identifies differentially regulated genes for each cluster in comparison to the ensemble of all cells:

```
> cdiff <- clustdiffgenes(sc,pvalue=.01)
```

It returns a list with a `data.frame` element for each cluster that contains the mean expression across all cells (`mean.all`) and in the cluster (`mean.cl`), the fold-change in the cluster versus all cells (`fc`), and the p-value for observing the difference in mean expression between the cluster and all cells based on binomial counting statistics (`pv`). To compute this probability the median total transcript count in each cell was rescaled to match the original median transcript count of all cells in the cluster. This step ensures that the p-value is not artificially inflated, e. g., due to an over-representation of cells with low transcript counts in a given cluster. Only genes with `pv` lower than pvalue are shown. The table is ordered by increasing `pv`, i. e. the most significantly up- or down-regulated genes will appear at the top of the list.

```
> head(cdiff$cl.3,10)
                 mean.all    mean.cl          fc           pv
Chgb__chr2      11.1210716 164.928066 14.83023147 0.000000e+00
Tac1__chr6       1.1824430  19.149883 16.19518556 0.000000e+00
Aldob__chr4     85.8161374  10.923715  0.12729209 2.420725e-25
Reg1__chr6      56.1138614   3.397515  0.06054680 1.087831e-20
Prap1__chr7     36.2228451   2.351300  0.06491208 1.185162e-13
Fth1__chr19     46.1119949   7.925666  0.17187863 8.727527e-13
Hopx__chr5       1.7432776  15.459437  8.86802932 6.686474e-11
Mt1__chr8       33.8109520   4.054829  0.11992651 1.196063e-10
Tph1__chr7       0.7284583   9.632583 13.22324553 5.959931e-09
Neurog3__chr10  1.0521867  10.639959 10.11223535 1.666593e-08
```

The tables with differentially expressed genes can be written to tab-separated text files:

```
> for ( n in names(cdiff) )
write.table(data.frame(GENEID=rownames(cdiff[[n]]), cdiff[[n]]), paste(
paste("cell_clust_diff_genes",sub("\\.","\\_",n), sep="_"), ".xls",
sep=""), row.names=FALSE, col.names=TRUE, sep="\t", quote=FALSE)
```

To specifically examine expression differences between two clusters or two sets of clusters, the function `diffgenes` can be used:

```
> d <- diffgenes(sc,cl1=1,cl2=c(2,3),mincount=5)
```

Input arguments are two vectors with sets cluster numbers, `cl1` and `cl2`, that should be compared, for instance, cluster 1 with the union of clusters 2 and 3. Only genes with more than `mincount` transcript in at least a single cell of `cl1` or `cl2` are evaluated.
The function computes the z-score for expression differences between the two groups using the average of the standard deviation in `cl1` and `cl2`. If `cl1` or `cl2` contains only a single cluster, the standard deviation is estimated by the squareroot of the mean. The function returns a list of five elements:

`z:`	a vector of z-scores in decreasing order with genes up-regulated in `cl1` appearing at the top of the list.
`cl1:`	a `data.frame` with expression values for cells in `cl1`.
`cl1n:`	a vector of cluster numbers in `cl1`.
`cl2:`	a `data.frame` with expression values for cells in `cl2`.
`cl2n:`	a vector of cluster numbers in `cl2`.

The function `plotdiffgenes` can be used to plot expression in the two groups for a given gene (Figure 8).

```
> plotdiffgenes(d ,gene="Aldob__chr4")
```

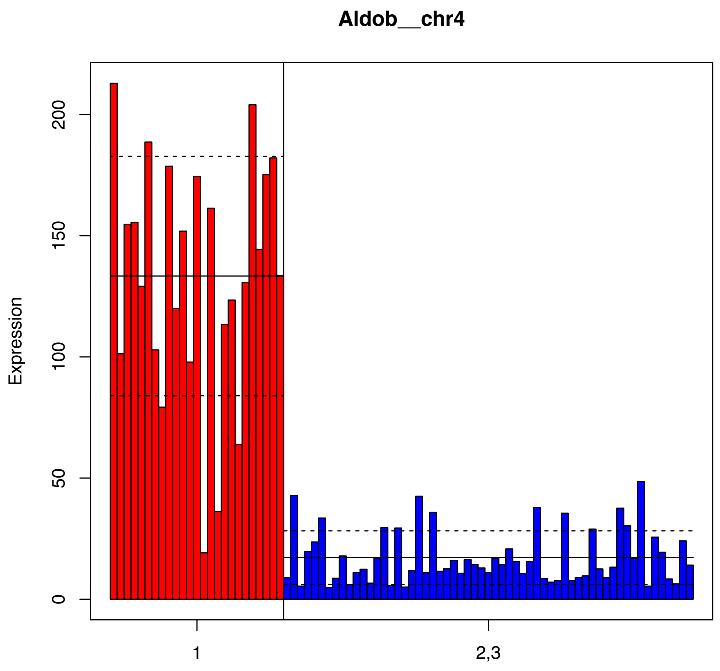

Figure 8. Expression profile of the gene *Aldob* in two sets of clusters (`cl1` and `cl2`) as computed by diffgenes. Cells in `cl1` and `cl2` are highlighted in red and blue, respectively. Mean expression and standard deviation are indicated by the black solid and broken lines, respectively.

The final clusters can also be plotted in a heatmap (Figure 9):

```
> clustheatmap(sc,final=TRUE,hmethod="single")
```

Input arguments are explained in paragraph II. To plot the final clusters instead of the k-means clustering, the logical argument `final` has to be set to `TRUE`.

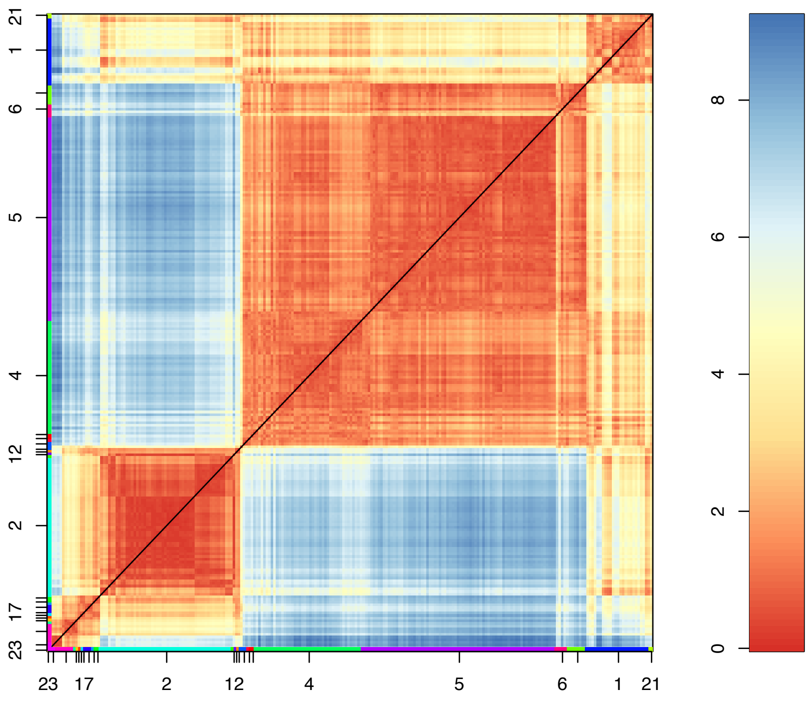

Figure 9. Heatmap of final clusters. Individual clusters are highlighted with rainbow colors along the x- and y-axes. Since not all cluster labels can be shown in the plot, the function `clustheatmap` returns a vector with all clusters in the same order as depicted in the heatmap.

VI. Using t-SNE maps for cluster examination

In order to visualize the partition of the cell population into clusters, the t-Distributed Stochastic Neighbor Embedding (t-SNE) algorithm[5] can be used, implemented in the R package `tsne`. A t-SNE map reduces the dimension of the input data to a low dimensional space, in our case to a two-dimensional space, and preserves all pairwise distances between the data-points as good as possible.

A t-SNE map is computed by the following command:

```
> sc <- comptsne(sc,rseed=15555)
```

The method is executed with a fixed seed for the random number generator to yield exactly reproducible maps across different runs. The seed can be varied by changing the numeric argument `rseed` (default is `15555`). A data.frame with the coordinates in the two dimensional output space is stored in the slot `sc@tsne`.

A t-SNE map can now be drawn for the original cluster and for the final clusters

Original clusters:

```
> plottsne(sc,final=FALSE)
```

Final clusters (Figure 10):

```
> plottsne(sc,final=TRUE)
```

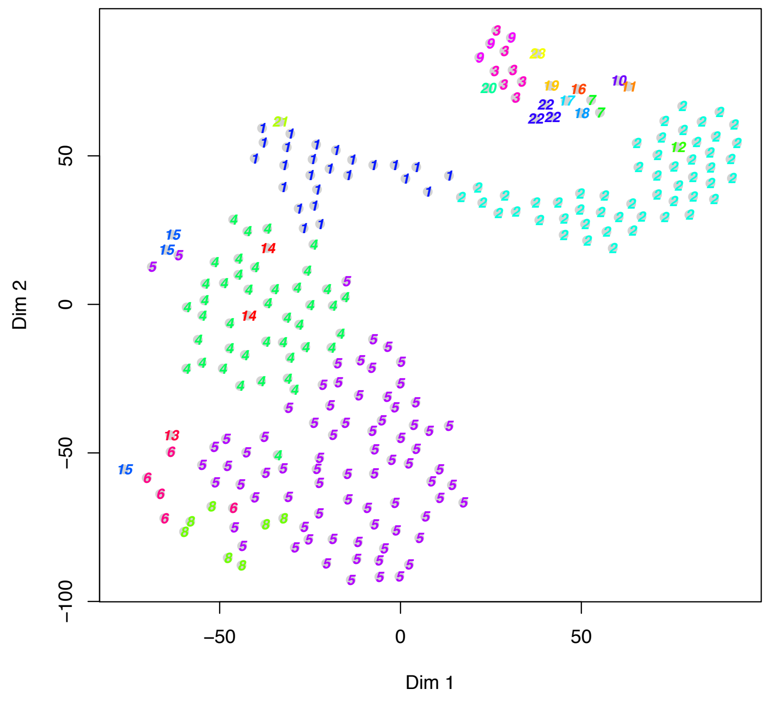

Figure 10. t-SNE map representation of all cells. All final clusters are highlighted in different colors and the cluster number is written on top of each cell.

The t-SNE map representation can also be used to analyze expression of a gene or a group of genes, to investigate cluster specific gene expression patterns (Figure 11):

```
> g <- c("Apoa1__chr9", "Apoa1bp__chr3", "Apoa2__chr1", "Apoa4__chr9",
"Apoa5__chr9")
> plotexptsne(sc,g)
```

The function requires a vector `g` of gene identifiers (or a single gene identifier) and highlights the aggregated expression level of these genes by superimposing a color scale on the t-SNE map. With the argument `n` a title of the plot can be specified, which by default corresponds to the first element of `g`. If the logical argument `logsc` is set to `TRUE` the log-transformed transcript counts are highlighted.

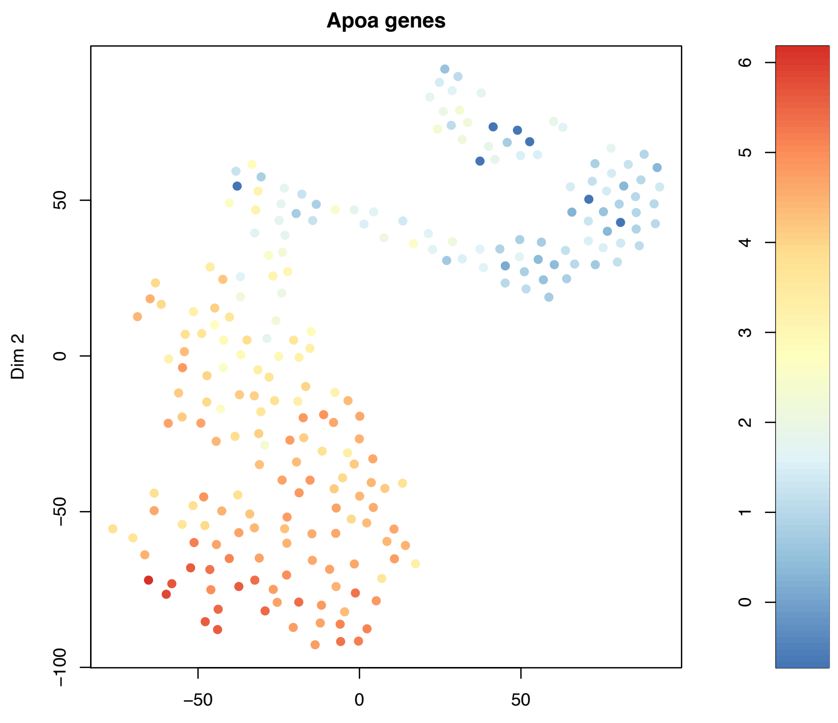

Figure 11. t-SNE map with color code representation of log-transformed gene expression aggregated across all Apoa genes.

The method `plotlabelstsne` allows plotting labels on top of each cell in the t-SNE map. By default, the column names of `sc@ndata`, i. e. the names of all single cells are plotted (Figure 12):

```
> plotlabelstsne(sc, labels=names(sc@ndata))
```

The argument labels can be any vector of labels with the same length as the column number of `sc@ndata`, `ncol(sc@ndata)`.

The method `plotsymbolstsne` can be used to label groups of cells in the t-SNE map by different symbols and colors (Figure 13).

```
> plotsymbolstsne(sc,types=sub("\\_\\d+$","", names(sc@ndata)))
```

For instance, cells from different replicates can be labeled if analyzed together. For this purpose a vector with common identifiers for all cells of a group has to be supplied as the `types` argument. For examples, if column names denote the experiment and a numeric identifier for the cell, a `types` vector can be created by a pattern substitution:

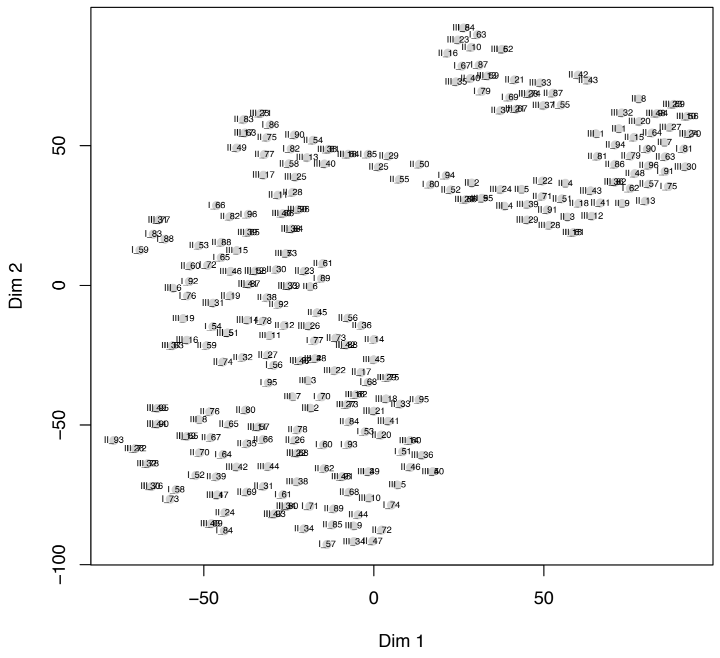

Figure 12. t-SNE map with labels corresponding to the column names of the transcript count data.

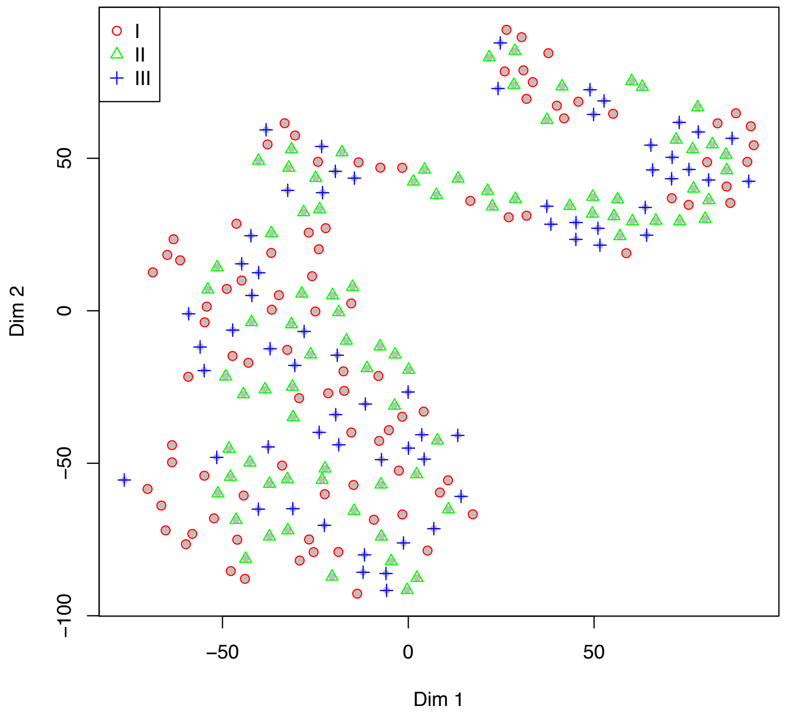

Figure 13. t-SNE map with symbols corresponding to different replicates analyzed together.

VII. Remarks on running RaceID

The RaceID algorithm can be run on single cell sequencing data produced by different techniques on material from arbitrary sources. Since the algorithm relies on statistics valid for absolute transcript

counts, the sequencing method should incorporate unique molecular identifiers. The performance for read based expression quantification has not bee tested.
In order to assess the performance of the method, the diagnostic plots have to be inspected carefully and it could be necessary to adjust parameter values.
It is crucial that the background model provides a reasonable fit to the data (Figure 5). If the variance of transcript counts is not a convex function of the mean (in logarithmic space) the background model will not yield a good fit and the algorithm cannot perform well. Moreover, it is assumed that a negative binomial provides a reasonable fit to the transcript count data for a given cell type and that expression of the majority of genes will not (strongly) depend on the cell type. All these assumption will most likely be valid for most systems and single cell sequencing techniques.
However, problems could arise if libraries with different complexities are combined and jointly analyzed by RaceID. In this case, it is strongly advised to eliminate cell-to-cell differences in technical noise by using downsampling instead of normalization in the `filterdata` method by setting the argument `downsample` to `TRUE`. This could also be necessary if the analyzed sample comprises cells with highly variable total transcript count (several orders of magnitude). It is always recommended to check consistency of the results with downsampling and median normalization. The minimum total transcript count `mintotal` as well as the minimum expression `minexpr` for a given number (`minnumber`) of cells should be chosen according to the sensitivity of the dataset. To obtain robust results `mintotal` should be at least of the order of ~1000. In general, only genes should be included, which are robustly expressed (>4 transcripts) in at least a single cell. More lowly expressed genes tend to introduce more noise and do not add statistical power. The parameter `maxepr` can be used to discard highly expressed genes that saturate the unique molecular identifiers (UMI). For these genes UMIs cannot be used to reliably infer transcript counts. The choice depends on the length of the UMI. For UMIs of length 4 genes with >500 transcripts per cell cannot be reliably quantified anymore[2].
The number of clusters in the initial k-means clustering step is determined based on gap statistic by default (Figure 1). However, the gap statistic can be noisy and a pronounced first local maximum might not exist. In this case, one should choose a number of clusters by setting `cln` to a number where the gap statistic starts saturating. The precise number will not be crucial, since the outlier identification step can introduce additional clusters. However, if the Jaccard's similarities (Figure 2) or the silhouette (Figure 3) of several clusters are low (Jaccard's similarity < 0.6 or silhouette < 0.1) the number of clusters should be lowered.
In the outlier identification step of the `findoutliers` method a crucial parameter is the probability threshold `probthr`, by default set to 10^{-3}. Visual inspection of the sensitivity plot (Figure 6) allows determining a reasonable value for this parameter. It should separate the broad tail from the initial steep decay of the distribution.
Additionally, the minimum number of outlier genes `outlg` can be varied. This parameter will also control the resolution. If only major cell types should be resolved that differ by a larger number of genes, `outlg` should be increased to higher numbers.
Another important parameter is the minimum transcript count `outminc` of a gene that has to be observed in at least a single cell in a cluster to include this gene in the statistical test. It will influence the size of the dataset (number of genes) and the multiple testing correction of the p-value. If a large number of outliers are detected, the stringency of the multiple testing correction can be increased by lowering this number.

For bug reports and any questions related to RaceID please email directly to dominic.gruen@gmail.com.

References

1. Jaitin, D. A. *et al.* Massively Parallel Single-Cell RNA-Seq for Marker-Free Decomposition of Tissues into Cell Types. *Science (80-.).* **343,** 776–779 (2014).

2. Grün, D., Kester, L. & van Oudenaarden, A. Validation of noise models for single-cell transcriptomics. *Nat. Methods* **11,** 637–40 (2014).

3. Islam, S. *et al.* Quantitative single-cell RNA-seq with unique molecular identifiers. *Nat. Methods* **11,** 163–6 (2014).

4. Tibshirani, R., Walther, G. & Hastie, T. Estimating the number of clusters in a data set via the gap statistic. *J. R. Stat. Soc. Ser. B (Statistical Methodol.* **63,** 411–423 (2001).

5. Van der Maaten, L. & Hinton, G. Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **9,** 2570–2605 (2008).