Reference manual:

FateID

Developed by Dominic Grün

Contents

1 Prerequisites

This manual explains how to run the FateID algorithm on sample data. FateID is a method for the quantification of cell fate bias in single cell transcriptome datasets comprising different cell types that emerge from a common progenitor. The progenitor populations are expected to be part of the dataset and the FateID algorithm was designed to learn a pre-existing bias of each progenitor cell to one or multiple alternative terminal fates. The strategy of the algorithm is to apply an iterative random forest classification[1] in order to quantify fate bias in increasingly naïve progenitors using cells that have been classified in previous iterations as training set.

To run FateID the R package can be directly installed from GitHub and loaded into the workspace:

For full functionality installation of Python v2.7 (or later) is required with the `numpy` and `sklearn` libraries. If Python or one of these libraries is not installed, the algorithm can still be run, but not all visualizations are possible (see below).

In the following sections we describe the application of FateID on sample data.

We do not explain all input and return arguments of the FateID functions in detail, since extensive description is available in the man pages for all of the functions.

2 Input data

The algorithm requires an expression data frame as input with cells as columns and genes as features. Column names are expected to correspond to cell IDs and row names are expected to correspond to gene IDs.

Example data are available from the FateID package. This dataset contains transcript counts of mouse intestinal epithelial cells positive for an *Lgr5*-lineage reporter after 5 days of lineage tracing[2], i. e. these cells are 5 days old progeny of *Lgr5*-positive intestinal stem cells:

```
> x <- intestine$x
> head(x[,1:5])
                         I5d_3     I5d_4     I5d_6    I5d_8      I5d_9
2200002D01Rik__chr7   3.751327 0.3486954 0.9355945 4.791691  2.4250484
2210407C18Rik__chr11  0.100000 0.5983680 0.3076682 0.100000  1.6469830
2310014L17Rik__chr7   0.100000 0.1000000 0.3076682 0.100000  0.1000000
Acsl5__chr19          1.132947 0.5983680 0.5161523 1.651566  0.8719749
Actb__chr5            4.813430 2.1173627 3.9683714 2.431937  4.7778317
Adh1__chr3           10.256359 7.1820558 0.3076682 7.179934 10.3793249
```

Moreover, FateID requires a partitioning of cells, which can be generated by any clustering method. For example the RaceID3 algorithm can be used to identify cell clusters, and a partitioning generated by this method is provided as part of the package. The partitioning has to be provided as a vector with integer values and component names corresponding to column names of the expression data frame:

```
> y <- intestine$y
> head(y)
 I5d_3  I5d_4  I5d_6  I5d_8  I5d_9 I5d_10
     2      3      4      2      5      4
```

3 Target clusters and feature selection

Cluster analysis can inform on the presence of mature cell types in the dataset, where cell types of distinct lineages correspond to different clusters (i. e. different numbers of the partition). In the

example data, cluster number 6 comprises enterocytes, marked by high expression of the *Alpi* gene, while cluster 9 represents mature Paneth cells (high expression of *Defa24*) and cluster 13 mature goblet cells (high expression of *Clca3*). Other rare cell types are only present in very low numbers and therefore were excluded from the analysis.

As further input for FateID, the endpoints of the differentiation trajectories, i. e. the most mature stages of all distinct cell lineages in the data set have to be defined by a vector of integer numbers representing the corresponding clusters in the partition `y`.

```
> tar <- c(6,9,13)
```

If a partitioning into cell types and states from a prior clustering analysis is not available, FateID can derive a partitioning based on marker gene information. For this strategy, a list of marker gene IDs is needed. Each component of this list contains one or more marker genes of a distinct lineage:

```
>  FMarker <- list(c("Defa20__chr8","Defa24__chr8"), "Clca3__chr3",
"Alpi__chr1")
>  xf <- getPart(x,FMarker,fthr=NULL,n=5)
> head(xf$part)
 I5d_3  I5d_4  I5d_6  I5d_8  I5d_9 I5d_10
     1      1      1      1      1      1
> head(xf$tar)
[1] 2 3 4
> tar <- xf$tar
> y <- xf$part
```

The `getPart` function extracts the top `n` cells expressing the markers of one of the lineages most highly and defines an expression threshold by the average expression across these cells. For this inference the expression level is aggregated across all markers of this lineage. The target cluster of a lineage is given by the ensemble of cells with aggregated marker gene expression higher than this threshold. Alternatively, a vector with threshold expression values for all lineages can also be provided as input argument `fthr`. The target cluster numbers will reflect order of the component of the FMarker list starting at 2. Cluster number 1 comprises all remaining cells that do not exhibit marker gene expression beyond the levels in `fthr`.

Once target clusters have been defined, FateID has the option to reclassify all remaining cells using the cells within the target clusters as input.

```
> rc <- reclassify(x, y, tar, clthr=.75, nbfactor=5, use.dist=FALSE,
seed=12345, nbtree=NULL, q=0.9)
> y  <- rc$part
```

The function returns a partition with the novel assignments after reclassification. The purpose of this step is to identify all cells with a pronounced bias towards one of the fates.
This step is optional but recommended to obtain larger training sets for random forests. This and other functions can be executed on expression data prior to or after feature selection. In the sample data, the data frame `x` contains only genes with variability exceeding a background level of combined technical and biological noise as inferred by RaceID3. Alternatively, the full data frame containing all genes can be used as input.

```
> v <- intestine$v
> rc <- reclassify(v, y, tar, clthr=.75, nbfactor=5, use.dist=FALSE,
seed=12345, nbtree=NULL, q=0.9)
> y  <- rc$part
```

The `reclassify` function also performs a feature selection based on importance sampling, i. e. all features with an importance larger than the `q`-quantile of the importance distribution for a given class are retained. The reduced expression table is returned by the function and can replace the original input expression data frame:

```
> x  <- rc$xf
```

Utilizing this function is recommended if the input data have not been subject to any other feature selection method.

Feature selection can also be performed utilizing a differential gene expression analysis.

```
> xf <- getFeat(v,y,tar,fpv=0.01)
```

This function compares gene expression within cells of a target cluster to the ensemble of all remaining cells and identifies genes that are significantly up-regulated in a target cluster with a p-value lower than `fpv`. The function returns a reduced expression data frame, which can be used for the subsequent analysis. In general, the reclassify function is more recommended for feature selection, since it reflects the information used for the random forest classification.

4 Computing the fate bias

The core function of FateID computes the fate bias for each cell in the dataset excluding cells within the target cluster. These cells are assigned to the lineage representing the respective target cluster with a probability of one and this probability does not change during inference of the fate bias of all other cells:

```
> fb <- fateBias(x, y, tar, z=NULL, minnr=5, minnrh=10, nbfactor=5, use.dist=FALSE, seed=12345, nbtree=NULL)
```

Apart from the (feature selected) expression data frame x, the partition y, and the vector of target clusters tar, the fateBias function takes further arguments as input. The optional argument z is a cell-to-cell distance matrix utilized to identify non-classified cells in the immediate neighborhood of all cells that have been classified as one of the target clusters in the previous iteration. By default this distance matrix will be computed as `z = 1 – cor(x)`, but if other distance measures are preferred a distance matrix can be provided by this argument.
The FateID algorithm computed by the `fateBias` function performs an iterative calculation. It starts with a set of cells representing each target cluster. For each target cluster, the `minnr` neighboring cells with the shortest median distance to all cells in the target cluster are extracted. The ensemble of the neighboring cells of all target clusters represents the test set of the next iteration. The `minnr` parameter therefore controls the step size of the algorithm. In each iteration, `minnr` cells times the number of target clusters are classified and can contribute to the training set in the next iteration.
The training set of this iteration comprises all cells assigned to one of the target clusters and the response vector is given by the partition of these cells. The classification of the test set is done based on the random forest votes: If a cell receives significantly more random forest votes for one target cluster versus all other clusters (based on sampling statistics with a p-value threshold of 0.05 it is assigned to this target cluster and contributes to the training set for the next iteration. All cells without a significant fate bias towards any one of the target clusters are not incorporated into the training set for the next iteration. However, the fraction of votes, which can be interpreted as a fate probability, is recorded and stored for all cells.
Another important parameter controls which cells contribute to the training set for a given iteration. At most `minnrh` cells from each target cluster contribute to the training set. These cells are selected as the `minnrh` cells with the shortest distance to any cell within the current training set. This parameter controls the gene expression horizon on the differentiation trajectory taken into account for the classification of the test set. If `minnrh` is set to `Inf` then all previously classified cells with a significant fate bias for one of the target clusters contribute to the training set. However, if gene expression changes follow complex dynamics along a differentiation trajectory, it can become detrimental to include very distant cells expressing maturation markers that are not expressed during the earliest stages of differentiation. It is generally advised to confine the expression horizon to smaller values in order to increase the specificity of the algorithm. However, the training set should still be large enough to warrant a confident classification. As a rule of thumb, the `minnrh` parameter should be set to 10 or larger values, depending on the size and coverage of the dataset. If the input data set is large and a large number of cells are available in the dataset for all lineages covering the entire differentiation trajectory, this parameter can be increased. We recommend testing the

robustness of the results to changes in `minnrh`. The step size `minnr` should be selected based on similar consideration. However, it is generally recommended to keep this number lower in order to avoid misclassification due to insufficient resolution. For the majority of datasets with several hundreds of cells we use `minnr=5` and `minnrh=10`.
As an alternative approach the FateID algorithm offers classification based on distances to all other cells. When `use.dist` is set to `TRUE`, then the distance matrix `z` (or `1 – cor(x)`) is interpreted as feature matrix.
The remaining arguments are control parameters of the random forest algorithm and usually do not have to be adjusted.
As outlined in the man packages of the `fateBias` function it returns a list of five components. The `votes` component is a data frame of random forest votes of all cells for each of the target clusters. The column names are given by a concatenation of a `t` with the number of the target cluster. The row names are given by cell IDs. The `probs` component has the same structure, but the votes for each cell are normalized to one in order to represent fate probabilities.
The `tr` component is a list of vectors. Each vector contains all cell IDs of cells with a significant fate bias towards the corresponding target cluster. Significant fate bias means significantly more votes for a given cluster than for any other cluster based on sampling statistics with a p-value below 0.05.
The fourth component is vector with all cell IDs ordered by random forest iteration in which they have been classified. The last component is a list of random forest objects produced by the function `randomForest` from the `randomForest` package object for all iterations.

5 Visualization of fate bias

Various dimensional reduction methods are commonly used for single cell transcriptome analysis in order to visually inspect the cell population structure. The FateID package computes a number of dimensional reduction representations to enable visualization of the fate bias and pseudo-temporal ordering by principal curve computation.

```
> dr <- compdr(x, z=NULL, k=c(2,3), lle.n=30, dm.sigma="local",
dm.distance="euclidean", tsne.perplexity=30, seed=12345)
```

The first two input parameters are the same as the ones to the `fateBias` function. The parameter k indicates the dimensions for which the dimensional reduction representations are computed. Typically, one wants to inspect data visually in 2 or 3 dimensions. However, it is possible to also compute dimensional reductions to a more than 3 dimensions and inspecting the data after projecting onto a subset of dimensions. The remaining parameters are main control parameter for the various algorithms used for dimensional reduction. The function performs computation of a t-SNE map[3] using the `Rtsne` package, classical multi-dimensional scaling using `cmdscale` from the `stats` package, locally linear embedding and modified locally linear embedding using python functions from the scikit learn open source Pyhton library, diffusion maps[4] from the `destiny` package, and a DDRTree graph representation[5] from the `DDRTree` representation as used by the Monocle2 algorithm[6].
The Python libraries `numpy` and `sklearn` have to be installed on the system. If Python or one of the libraries is not installed, the locally linear embedding will be computed in R using the `lle` package, but the computation of the modified locally linear embedding will be skipped.

All results can be plotted by the `plotFateMap` function.

The dimensional reduction representation with a highlighting of the partition can be plotted for any of the dimensional reduction algorithms in any of the dimensions computed by `compdr`. For example, a t-SNE map can be plotted in two dimensions (Figure 1):

```
> plotFateMap(y,dr,k=2,m="tsne")
```

Plotting in three dimensions opens an interactive RGL device to allow rotation of the plot and zooming in and out (Figure 1):

```
> plotFateMap(y,dr,k=3,m="tsne")
```

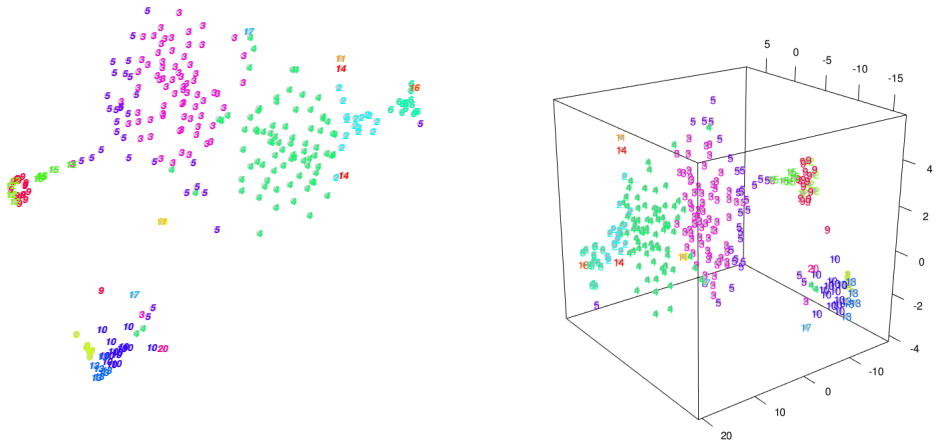

Figure 1: t-SNE representation of the input data in two dimensions (left) or three dimensions (right). Different clusters as defined by the partition y are highlighted in distinct colors.

The fate bias can now be highlighted in the dimensional reduction representation by providing the name of the target cluster additional argument (concatenation of a `t` and the cluster number) and the `fateBias` output (see Figure 2):

```
> plotFateMap(y,dr,k=2,m="tsne",fb=fb,g="t6")
```

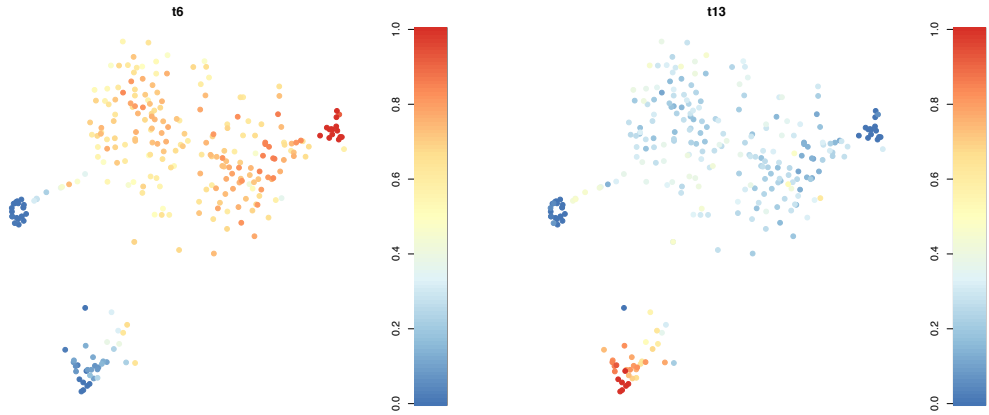

Figure 2: Representation of fate bias by color-coding of a dimensional reduction representation. In this example, the fate bias towards enterocytes (cluster 6) and goblet cells (cluster 13) is shown in a t-SNE map on the left and right, respectively.

To approximate the differentiation trajectories principal curves can be computed in the dimensionally reduced space. For the computation of a principal curve of a given target cluster, only those cells contribute that have a fraction of random forest votes for this target cluster larger than `trthr`. If this argument is not given only cells contribute with a significant bias for this target cluster in comparison to all others with a p-value < 0.05 derived from sampling statistics. The principal curves are plotted into the maps and returned by the function if the argument `prc` is `TRUE`.

```
> pr <- plotFateMap(y,dr,k=2,m="tsne",trthr=.33,fb=fb,prc=TRUE)
```

See Figure 3.

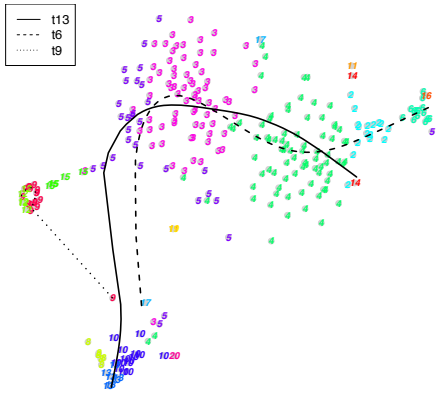

Figure 3: Principal curves computed for trajectories to three target clusters in the t-SNE map.

If the most naïve cluster representing a common upstream progenitor of all cell types is known, the number of this cluster can also be given as input argument `start` in order to initialize the principal curve computation with principal curves starting in the `start` cluster and ending in one of the target clusters. To enforce that principal curves traverse through the cell population representing the respective target cluster, the weight of cells within the initial target clusters is increased 10-fold for the principal curve computation.
If principal curves are computed, the return object of `plotFateMap` is a list of two objects. The first argument `pr` is itself a list of principal curve objects for each target cluster as computed by the `principal.curve` function from the `princurve` package. The second argument `trc` is a list of vectors with cell IDs for the principal curve of each target cluster containing the IDs of all cells with a significant fate bias or a fate bias greater than `trthr`, respectively. The cell IDs are ordered by their position along the principal curve as obtained from the point of their shortest distance to the curve.

The expression of a gene, or the aggregated expression across a group of genes, can also be highlighted in the dimensional reduction representation by providing a vector with the respective gene IDs as additional parameter `g`, an optional title `n` for the plot, and an expression data data frame `x` (see Figure 4):

```
> pr <-plotFateMap(y, dr, k=2, m="tsne", g=c("Defa20__chr8",
"Defa24__chr8"), n="Defa", x=v)
```

Finally, if `g` equals `"E"` and a `fateBias` object is provided as input the function will plot the entropy of the fate bias based on the probabilities for the different target clusters (see Figure 4). An entropy level of the fate bias will indicate the level of multipotency with larger fate bias entropies corresponding to more multipotent cell states. If executed with `g="E"`, the function will return a vector with the fate bias entropy of each cell (see Figure 4):

```
> E <- plotFateMap(y,dr,k=2,m="tsne",g="E",fb=fb)
> head(E)
    I5d_3     I5d_4     I5d_6     I5d_8     I5d_9    I5d_10
0.6202877 0.7095137 0.5689359 0.6983464 0.9179618 0.3958605
```

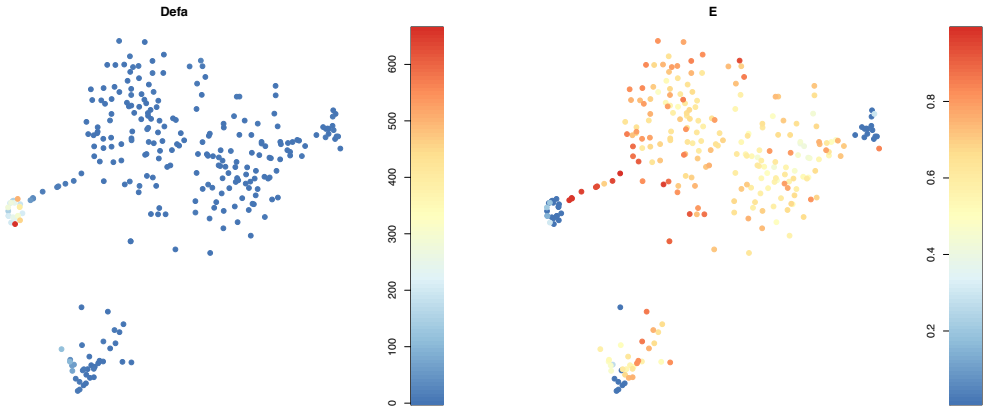

Figure 4: Expression aggregated across the two genes *Defa20* and *Defa24* (left) and the fate bias entropy (right) are highlighted in the dimensional reduction representation (left).

The principal curve can also be computed directly without calling the `plotFateMap` function:

```
> pr  <- prcurve(y,fb,dr,k=2,m="tsne",trthr=0.4,start=3)
```

This function has the same input arguments as `plotFateMap` and is invoked by the `plotFateMap` function.

6 Inspecting pseudo-temporal gene expression changes

FateID also provides functions for the visualization and analysis of pseudo-temporal gene expression changes.
For this purpose, cells with a fate bias towards a target cluster can be extracted. The principal curve analysis returns all cells along a differentiation trajectory in pseudo-temporal order. For example, cells with a fate bias towards cluster 6 in pseudo-temporal order can be extracted by the following command:

```
> n <- pr$trc[["t6"]]
```

Alternatively, other published methods for pseudo-temporal ordering, e. g., Monocle2[6] or diffusion pseudo-time[7], can be used after selecting cells by cell fate probability.
As one option, we implemented a function that performs filtering by cell fate probability and subsequent pseudo-temporal ordering by diffusion pseudo-time:

```
> trc <- dptTraj(x,y,fb,trthr=.25,distance="euclidean",sigma=1000)
```

If a threshold `trthr` for the minimum required cell fate probability is not given, than only cells with a significant fate bias towards a particular target cluster are used to populate the respective trajectory. The function returns a list of vectors for each target cluster containing all cells on the inferred differentiation trajectory to this cluster in pseudo-temporal order.

In the second step, the input expression data are defined:

```
> fs  <- filterset(v,n=n,minexpr=2,minnumber=1)
```

Here, we are using the full expression data frame `v` without feature selection.
The second argument is the vector of cell ids corresponding to valid row names of the expression table in pseudo-temporal order.

The `filterset` function can be used to eliminate lowly expressed genes on the trajectory from the subsequent analysis and has two additional arguments by discarding genes, which are not expressed at a level of `minexpr` or higher in at least `minnumber` of cells. The function returns a filtered expression data frame with genes as rows and cells as columns in the same order as in the input vector `n`.
In the third step, a self-organizing map (SOM) of the pseudo-temporal expression profiles is computed:

```
> s1d <- getsom(fs,nb=1000,k=5,locreg=TRUE,alpha=.5)
```

This map provides a grouping of similar expression profiles into modules. The first input argument is again an expression data frame. In this case, we use the filtered expression table generated by the `filterset` function to retain only genes that are expressed on the trajectory under consideration.
Pseudo-temporal expression profiles along the differentiation trajectory of interest are computed after smoothening either by running mean of window size `k` or, if `locreg` is `TRUE` local regression with smoothing parameter `alpha`.

This function returns a list of the following three components, i. e. a `som` object returned by the function `som` of the package `som`, a data frame `x` with smoothened and normalized expression profiles, and a data frame `zs` of z-score transformed pseudo-temporal expression profiles.

The SOM is then processed by another function to group the nodes of the SOM into larger modules and to produce additional z-score transformed and binned expression data frames for display:

```
> ps  <- procsom(s1d,corthr=.85,minsom=3)
```

The first input argument is given by the SOM computed by the function `getsom`. The function has two additional input parameters to control the grouping of the SOM nodes into larger modules. The parameter `corthr` defines a correlation threshold. If the correlation of the z-scores of the average normalized pseudo-temporal expression profiles of neighboring nodes in the SOM exceeds this threshold genes of the neighboring nodes are merged into a larger module. Only modules with at least `minsom` genes are kept. The function returns a list of various data frames with normalized, z-score-transformed, or binned expression along with the assignment of genes to modules of the SOM (see man pages for details).

The output of the processed SOM can be plotted using the `plotheatmap` function.
First, in order to highlight the clustering partition y the same color scheme as used in `plotFateMap` can be generated (alternatively, both functions can be executed with an arbitrary user-defined scheme):

```
> set.seed(111111)
> fcol <- sample(rainbow(max(y)))
```

Now, the different output data frames of the `procsom` function can be plotted.

Plot average z-score for all modules derived from the SOM:

```
> plotheatmap(ps$nodes.z, xpart=y[n], xcol=fcol, ypart=unique(ps$nodes),
xgrid=FALSE, ygrid=TRUE, xlab=FALSE)
```

Plot z-score profile of each gene ordered by SOM modules:

```
> plotheatmap(ps$all.z, xpart=y[n], xcol=fcol, ypart=ps$nodes, xgrid=FALSE,
ygrid=TRUE, xlab=FALSE)
```

Plot normalized expression profile of each gene ordered by SOM modules:

```
> plotheatmap(ps$all.e, xpart=y[n], xcol=fcol, ypart=ps$nodes, xgrid=FALSE,
ygrid=TRUE, xlab=FALSE)
```

Plot binarized expression profile of each gene (z-score < -1, -1 < z-score < 1, z-score > 1):

```
> plotheatmap(ps$all.b, xpart=y[n], xcol=fcol, ypart=ps$nodes, xgrid=FALSE,
ygrid=TRUE, xlab=FALSE)
```

All plots are shown in Figure 5.

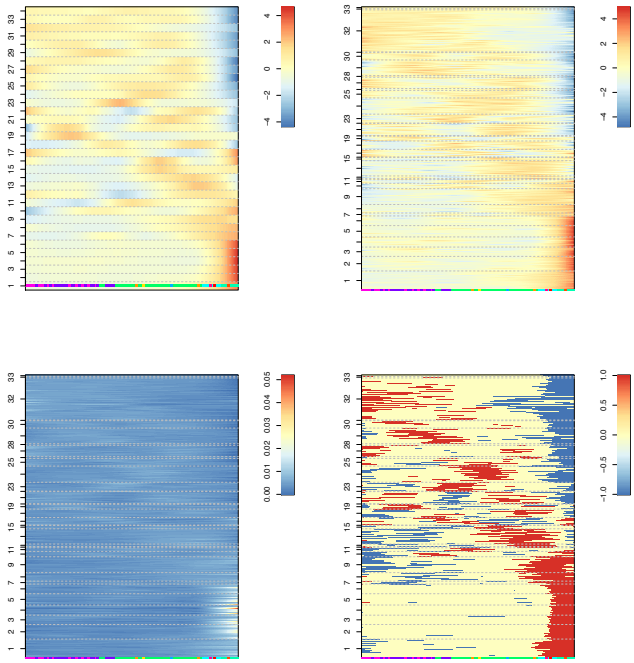

Figure 5: Heatmaps of expression profiles ordered by SOM-derived expression modules. The profiles are shown for the mean z-score of all modules (upper left), for all individual genes in these modules (upper right), for the normalized expression of individual genes (lower left), and for binned expression of all genes based on the z-score (lower right).

In order to inspect genes within individual modules of the SOM these genes can be extracted given the number of the module. The module numbers are contained in the return argument `nodes` of the `procsom` function and can be extracted, e. g. for module number 1:

```
> g <- names(ps$nodes)[ps$nodes == 1]
```

The average pseudo-temporal expression profile of this group can be plotted by the function `plotexpression` (see Figure 6):

```
> plotexpression(fs, y, g, n, k=25, col=fcol, name="Node 1", cluster=FALSE,
locreg=TRUE, alpha=.5, types=NULL)
```

In the same way it is possible to plot expression profiles of individual genes, e. g.:

```
> plotexpression(fs, y, "Clca4__chr3", n, k=25, col=fcol, cluster=FALSE,
locreg=TRUE, alpha=.5, types=NULL)
```

See Figure 26. It is also possible to highlight the data points as specific symbols, for example reflecting batches, by using the `types` argument (see Figure 26):

```
> plotexpression(fs, y, g, n, k=25, col=fcol, name="Node 1", cluster=FALSE,
locreg=TRUE, alpha=.5, types=sub("\\_\\d+","",n))
```

See man pages for detailed explanation of the input parameters.

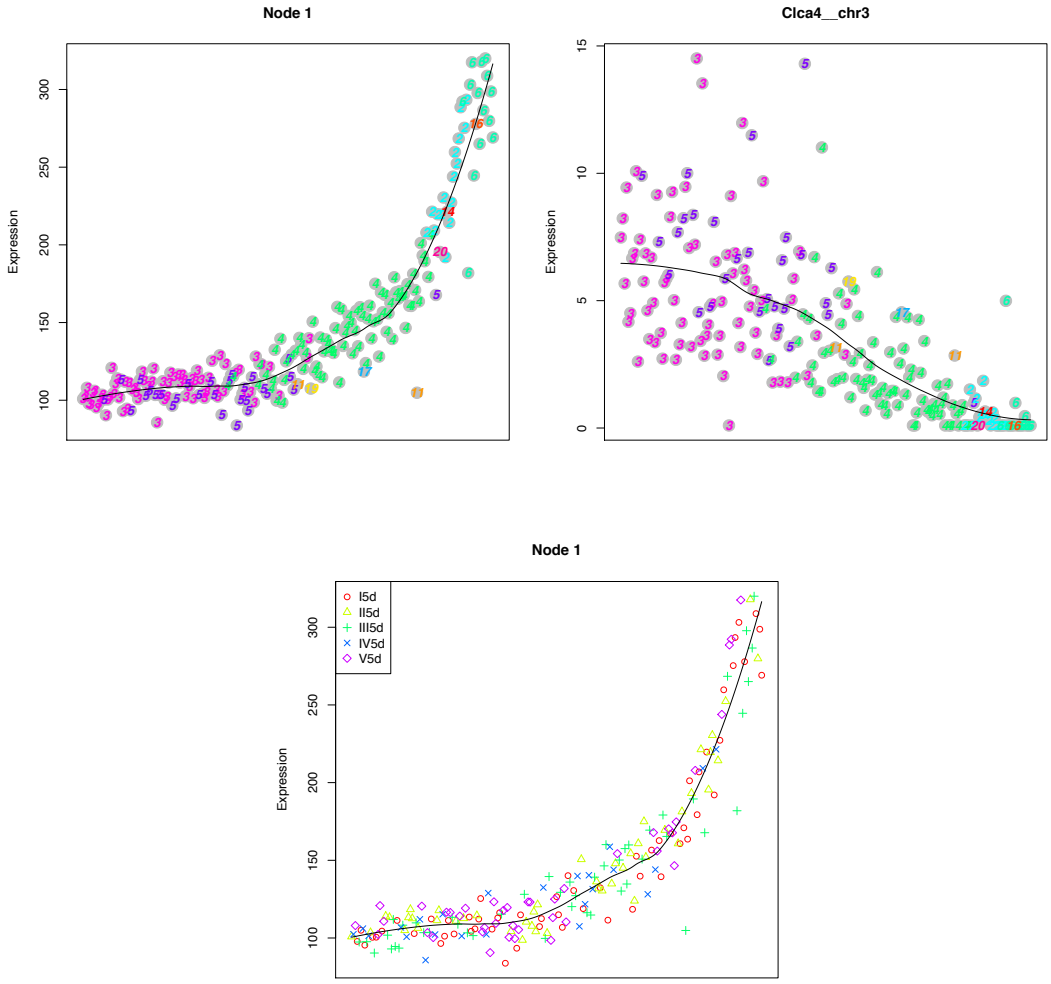

Figure 6: Pseudo-temporal expression profiles of all genes in module 1 (left) and the stem cell marker Clca4 (right). The bottom panel shows the expression profile of module 1 genes, but highlight the batch origin of each cell.

7 Differential gene expression analysis

Together with FateID a function for differential gene expression analysis is provided. This function implements an approach similar to DESeq[8] by which negative binomial distributions are fitted to transcript counts in two populations, representing, for instance, two different cell types. By default, the function should be executed on normalized data and estimates the dispersion parameter from the polynomial fit of the log-transformed transcript count variance as a function of the log-transformed mean expression. This is done on the input data, either after pooling cells from the two populations or on each of the populations separately. Alternatively, one can also provide a function for the variance-mean dependence as an input parameter, e. g. the dependence computed in the RaceID2[2] outlier identification. As another option, the function can also execute a standard DESeq2[9] differential expression analysis.

To run the analysis, an expression data frame is needed as input together with two vectors of cell IDs corresponding to subsets of column names of this data frame.

For example, this function could be used to identify genes differentially expressed between cells in the original clusters 3,4,5 (the progenitor compartment) that are either biased towards enterocytes (cluster 6) or goblet cells (cluster 13) with a fate-probability of >0.5:

```
> thr <- .5
> a   <- "t13"
> b   <- "t6"
> cl  <- c(3,4,5)
> A <- rownames(fb$probs)[fb$probs[,a] > thr]
> A <- A[y[A] %in% cl]
> B <- rownames(fb$probs)[fb$probs[,b] > thr]
> B <- B[y[B] %in% cl]
> de <- diffexpnb(v,A=A,B=B,DESeq=FALSE,norm=F,vfit=NULL,locreg=FALSE)
```

Here, we use the full expression data frame without feature selection as input. Normalization is not performed, since `v` is already normalized by size. With the chosen input parameters, `diffexpnb` will infer the dispersion parameter of the negative binomial transcript level distributions for the two populations based on a variance-mean dependence derived as in RaceID2[2]. Alternatively, a local regression can be utilized, if `locreg` equals `TRUE`. Another option is to run a DESeq2[9] analysis, if DESeq is `TRUE`.
The function returns a list of three components. The results of the differential gene expression analysis are stored in the `res` component, which is a data frame displaying mean expression of the two sets, fold change and log2 fold change between the two sets, the p-value for differential expression (`pval`) and the Benjamini-Hochberg corrected false discovery rate (`padj`).

The results can be plotted by the following function:

```
> plotdiffgenesnb(de,mthr=-4,lthr=0,Aname=a,Bname=b,padj=F)
```

See Figure 7.

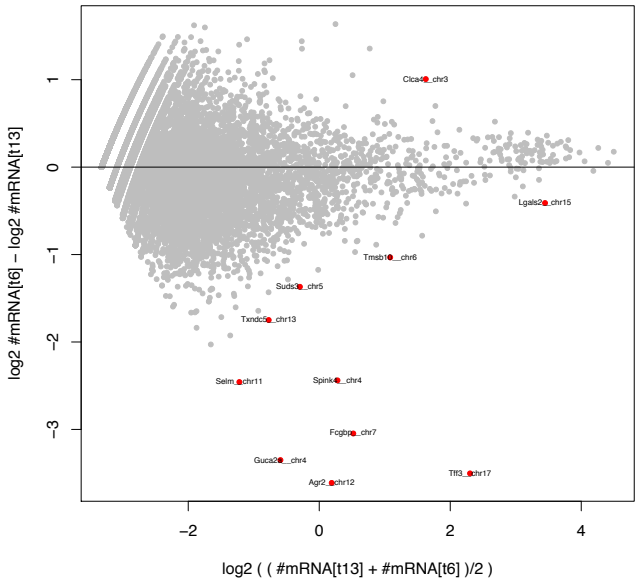

Figure 7: MA plot of differentially expressed genes between cells in clusters 3, 4 and 5 that exhibit a fate probability >0.5 either towards clusters 13 or cluster 6.

8 Inspecting fate bias of gene expression

To enable further inspection of fate-specific marker gene expression, FateID provides a function to plot the expression levels of two genes against each other in a scatter plot. These genes could, for instance, represent markers of two different lineages, and the function reveals whether the expression domains of these genes fall into distinct clusters or whether they correlate with the fate bias towards a particular lineage.
To visualize a correlation of gene expression domain and cluster origin of the cells, the `gene2gene` function can be called with a gene expression data frame and a clustering partition as input (see Figure 8):

```
> gene2gene(intestine$v,intestine$y,"Muc2__chr7","Apoa1__chr9")
```

If the fate bias towards a particular target cluster, e. g. cluster 6, should be highlighted, the output `fb` of the `fateBias` function and the column name of `fb$probs` corresponding to the target cluster, i. e. `t6`, are required as additional input parameters:

```
> gene2gene(intestine$v, intestine$y, "Muc2__chr7", "Apoa1__chr9", fb=fb,
tn="t6", plotnum=F)
```

See man pages for additional input arguments.

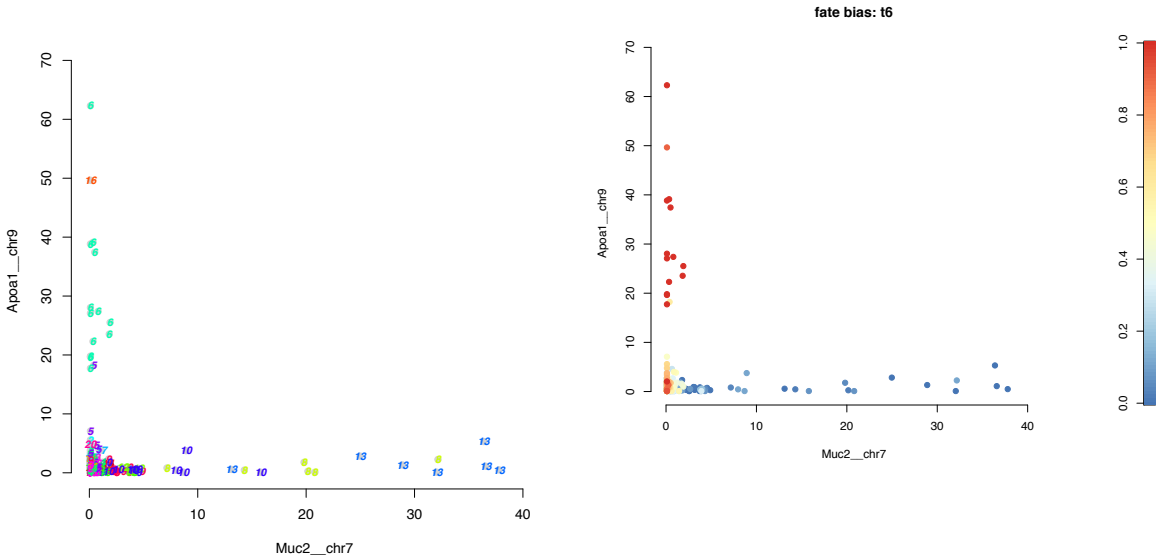

Figure 8: Scatterplot for the comparison of the expression domains of two genes highlighting the cluster of origin for each cell (left panel) or the fate bias towards a particular target cluster.

9 Extracting genes with high importance for classification

To identify genes involved in early priming and fate decision at all stages of differentiation, it is informative to extract all the genes with a high importance value (see details of the `randomForest` function in the `randomForest` package) for the classification into a given target cluster in at least one random forest iteration:

```
> k <- impGenes(fb,"t10")
```

This function requires the output object of the `fateBias` function and the number of a target clusters, concatenated with a `t` as input. Additional optional arguments specify cutoffs for the importance values (see man pages). It returns a list of two objects, one data frame with mean importance values for all genes that passed the thresholds in at least of the iterations as rows and the iterations as columns, and a corresponding data frame with the standard deviation of the importance values.

The function also plots a heatmap of the importance values with hierarchical clustering of the genes (see Figure 9).

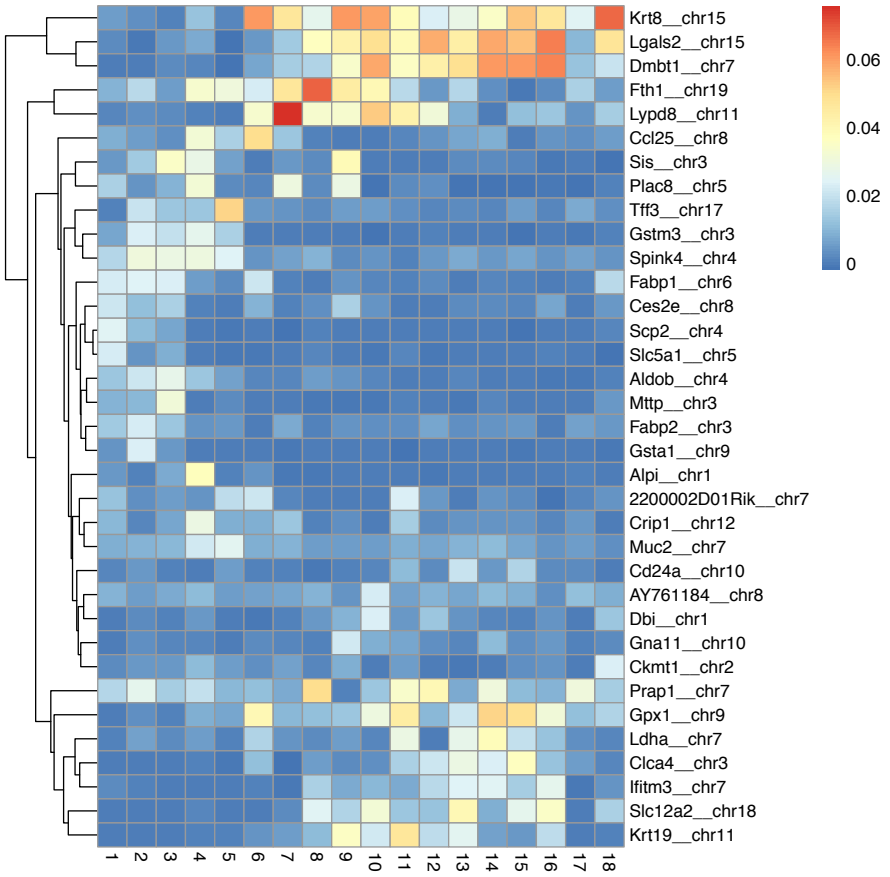

Figure 9: Heatmap of importance values of genes with high importance for classifying into cluster 6.

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

References

1. Breiman, L. & Leo. Random Forests. *Mach. Learn.* **45,** 5–32 (2001).
2. Grün, D. *et al.* De Novo Prediction of Stem Cell Identity using Single-Cell Transcriptome Data. *Cell Stem Cell* **19,** 266–277 (2016).
3. van der Maaten, L. & Hinton, G. Visualizing Data using t-SNE. *J. Mach. Learn. Res.* **9,** 2570–2605 (2008).
4. Haghverdi, L., Buettner, F. & Theis, F. J. Diffusion maps for high-dimensional single-cell analysis of differentiation data. *Bioinformatics* **31,** 2989–98 (2015).
5. Mao, Q., Wang, L., Goodison, S. & Sun, Y. Dimensionality Reduction Via Graph Structure Learning. in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '15* 765–774 (ACM Press, 2015). doi:10.1145/2783258.2783309
6. Qiu, X. *et al.* Single-cell mRNA quantification and differential analysis with Census. *Nat.*

Methods **14,** 309–315 (2017).

7. Haghverdi, L., Büttner, M., Wolf, F. A., Buettner, F. & Theis, F. J. Diffusion pseudotime robustly reconstructs lineage branching. *Nat. Methods* **13,** 845–848 (2016).
8. Anders, S. & Huber, W. Differential expression analysis for sequence count data. *Genome Biol.* **11,** R106 (2010).
9. Love, M. I., Huber, W. & Anders, S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.* **15,** 550 (2014).