Analyzing 10X Chromium single-cell RNA-seq profiles of mouse mammary gland

Yunshun Chen [1,2]

[1]The Walter and Eliza Hall Institute of Medical Research, 1G Royal Parade, Parkville, Victoria 3052, Australia
[2]Department of Medical Biology, The University of Melbourne, Victoria 3010, Australia

December 6, 2018

Contents

1 Overview . . . 3

2 Preliminary analysis . . . 3
- 2.1 Pre-processing the raw data . . . 3
- 2.2 Downloading the read counts . . . 3
- 2.3 Reading in the data . . . 3
- 2.4 Gene annotation . . . 4
- 2.5 Quality control . . . 5

3 Exploratory data analysis . . . 6
- 3.1 Creating a `SingleCellExperiment` object . . . 6
- 3.2 Normalization . . . 7
- 3.3 Modelling the mean-variance trend . . . 8
- 3.4 Principal component analysis . . . 9
- 3.5 t-SNE visualization . . . 11
- 3.6 Cell clustering . . . 12
- 3.7 Marker gene detection . . . 13

4 Correlating with bulk RNA-seq profiles . . . 16
- 4.1 Gene signatures from a previous study . . . 16
- 4.2 Ternary plot visualization . . . 16
- 4.3 Signature scores . . . 18

5 Diffusion analysis . . . 19
- 5.1 Create a `CellDataSet` object . . . 19
- 5.2 Dispersion estimation . . . 20

5.3 Ordering cells in pseudotime 20

6 Data and software availability 22

1 Overview

Single-cell RNA sequencing (scRNA-seq) has become a widely used technique that allows researchers to profile the gene expression and study molecular biology at the cellular level. It provides biological resolution that cannot be achieved with conventional bulk RNA-seq experiments on cell populations. Of all the existing platforms the 10X Genomics Chromium system has become the most popular and commonly used one. Each run of 10X generates millions of UMI counts across thousands of cells.

Here, we provide a detailed workflow for analyzing a 10X Chromium single-cell RNA-seq data. The 10X data we use here is from a study of mouse epithelial mammary gland [1]. It contains gene expression profiles of over 3000 adult total epithelial cells.

2 Preliminary analysis

2.1 Pre-processing the raw data

The raw 10X data usually come in BCL format. They need to be pre-processed by software tools such as *cellranger* (a software developed by the 10X Genomics company itself).

For this particular data, we used *cellranger* to convert BCL files to the FASTQ format, align the reads to the mouse genome and quantify the UMI counts for each gene in each cell. The entire analysis in this workflow is conducted within the R environment using the outputs from *cellranger*. We do not cover the details of running *cellranger* as they are beyond the scope of this workflow. The information of how to run *cellranger* is available at https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/what-is-cell-ranger.

2.2 Downloading the read counts

The *cellranger* outputs of this data are publicly available at https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSM2510617. It contains three data files: a sparse matrix object `GSM2510617_P7-matrix.mtx.gz`, cell barcodes `GSM2510617_P7-barcodes.tsv.gz` and the gene annotation `GSM2510617_P7-genes.tsv.gz`.

Here we create a subfolder named `Data` under the current working directory. We download the three files and store them in the `Data` directory.

```
> dir("Data/")
```

```
[1] "BulkSignatures.RData"          "GSM2510617_P7-barcodes.tsv.gz"
[3] "GSM2510617_P7-genes.tsv.gz"    "GSM2510617_P7-matrix.mtx.gz"
[5] "Mus_musculus.gene_info.gz"
```

2.3 Reading in the data

We load in the count matrix using the `read10X()` function from the *edgeR* package. This will create a `DGEList` object where each row corresponds to a gene and each column corresponds to a cell.

```
> library(scater)
> library(scran)
> library(edgeR)
> options(digits=3)
> dge <- read10X(
+     mtx="GSM2510617_P7-matrix.mtx.gz",
+     genes="GSM2510617_P7-genes.tsv.gz",
+     barcodes="GSM2510617_P7-barcodes.tsv.gz",
+     path="Data",
+     DGEList=TRUE)
```

The dimension of the `DGEList` object can be viewed as follows.

```
> dim(dge)
```

```
[1] 27998  3302
```

2.4 Gene annotation

The *cellranger* annotates genes with both ENSEMBL identifiers and gene symbols. However, the gene symbols contain aliases which are not the official symbols of the genes.

```
> head(dge$genes)
```

```
                    Symbol
ENSMUSG00000051951    Xkr4
ENSMUSG00000089699  Gm1992
ENSMUSG00000102343 Gm37381
ENSMUSG00000025900     Rp1
ENSMUSG00000109048     Rp1
ENSMUSG00000025902   Sox17
```

Here, we use the NCBI refSeq annotation to convert gene aliases to official symbols. The annotation file can be downloaded from https://ftp.ncbi.nih.gov/gene/DATA/GENE_INFO/Mammalia/Mus_musculus.gene_info.gz. We add the official symbol and Entrez gene ids of the mouse genes into the annotation.

```
> ann <- alias2SymbolUsingNCBI(dge$genes$Symbol, required.columns=c("GeneID",
+     "Symbol"), gene.info.file="Data/Mus_musculus.gene_info.gz")
> dge$genes <- cbind(dge$genes, Official=ann$Symbol, GeneID=ann$GeneID)
> head(dge$genes)
```

```
                    Symbol Official    GeneID
ENSMUSG00000051951    Xkr4     Xkr4    497097
ENSMUSG00000089699  Gm1992   Gm1992 100038975
ENSMUSG00000102343 Gm37381     <NA>      <NA>
ENSMUSG00000025900     Rp1      Rp1     19888
ENSMUSG00000109048     Rp1      Rp1     19888
ENSMUSG00000025902   Sox17    Sox17     20671
```

2.5 Quality control

Quality control is essential for 10X data. Cells of bad quality and genes of low expression shall be removed prior to the analysis.

Two common measures of cell quality are the library size and the number of expressed genes in each cell. Another measure to look at is the proportion of reads from mitochondrial genes in each cell. High expression of mitochondrial genes indicates bad quality of the cell. Here, we calculate the percentage of reads from mitochondrial genes and the number of expressed genes (with at least one read) in each cell.

```
> mito <- grep("^mt-", dge$genes$Symbol)
> percent.mito <- colSums(dge$counts[mito, ]) / dge$samples$lib.size
> nGenes <- colSums(dge$counts != 0)
> dge$samples <- cbind(dge$samples, percent.mito=percent.mito, nGenes=nGenes)
> head(dge$samples, n=10)
```

	group	lib.size	norm.factors	Barcode	percent.mito	nGenes
1	1	4117	1	AAACATACCCTTTA-1	0.012145	1687
2	1	1220	1	AAACATACGAACTC-1	0.001639	676
3	1	3965	1	AAACATACTGCAAC-1	0.000757	1429
4	1	2551	1	AAACGCACTTCTCA-1	0.000392	1292
5	1	1858	1	AAACGCTGAAGCCT-1	0.009150	941
6	1	7293	1	AAACGCTGACCTTT-1	0.001508	2324
7	1	2145	1	AAACGCTGGATACC-1	0.003730	1044
8	1	6001	1	AAACGGCTAACGTC-1	0.007499	2060
9	1	1851	1	AAACGGCTCTTCCG-1	0.002701	987
10	1	1696	1	AAACGGCTGAACCT-1	0.002948	931

Scatter plots are produced for visualization (Figure 1). It can be seen that all the cells have low proportion of reads from mitochondrial genes ($< 5\%$). Hence, we keep all the cells in the analysis.

```
> par(mfrow=c(1,2))
> plot(dge$samples[,c("lib.size","nGenes")], pch=16, cex=0.7)
> plot(dge$samples[,c("lib.size","percent.mito")], pch=16, cex=0.7)
```

Then we perform gene filtering. We filter out genes expressed in fewer than 1% of the total cells. We remove the genes with no valid official symbols. We also remove duplicated genes if any.

```
> o <- order(rowSums(dge$counts), decreasing=TRUE)
> dge <- dge[o, ]
> keep1 <- rowSums(dge$counts > 0) >= ncol(dge)*0.01
> keep2 <- !is.na(dge$genes$Official)
> keep3 <- !duplicated(dge$genes$Official)
> keep <- keep1 & keep2 & keep3
> table(keep)
```

```
keep
FALSE  TRUE
18379  9619
```

```
> dge <- dge[keep1 & keep2 & keep3, ]
```

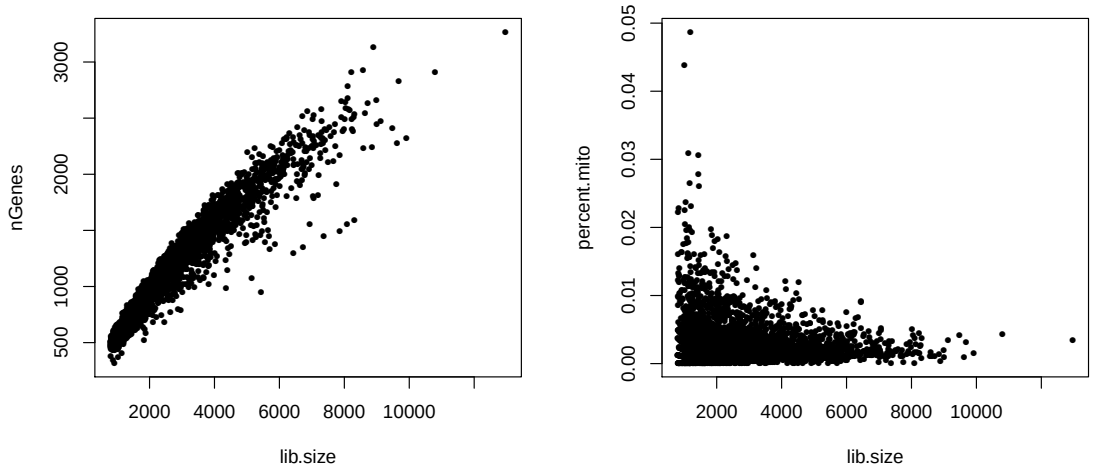

Figure 1: Scatter plots of numbers of expressed genes and proportion of mitochondrial reads against the library size
It confirms that the data is of good quality as there are decent number of genes expressed and the proportion of mitochondrial reads is low in each cell.

```
> rownames(dge) <- dge$genes$Official
```

3 Exploratory data analysis

3.1 Creating a `SingleCellExperiment` object

In this workflow, we use the *scran* package for the exploratory analysis of the 10X data. *scran* works on a `SingleCellExperiment` class object which is a light-weight container for single-cell genomics data.

We first create a `SingleCellExperiment` object using the count matrix as follows.

```
> sce <- SingleCellExperiment(list(counts=dge$counts))
> sce

class: SingleCellExperiment
dim: 9619 3302
metadata(0):
assays(1): counts
rownames(9619): Trf Spp1 ... Ubiad1 Zbtb22
rowData names(0):
colnames(3302): 1 2 ... 3301 3302
colData names(0):
reducedDimNames(0):
spikeNames(0):
```

3.2 Normalization

Normalization is useful for removing cell-specific biases. We apply the deconvolution method [2] to compute size factors for all cells. We perform some pre-clustering to break up obvious clusters and avoid pooling cells that are very different.

```
> clst <- quickCluster(sce, method="igraph", min.mean=0.5)
> table(clst)

clst
   1    2    3
1203 1251  848

> sce <- computeSumFactors(sce, cluster=clst)
> summary(sizeFactors(sce))

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.15    0.56    0.86    1.00    1.31    5.38
```

The size factors are strongly correlated with the library sizes, which indicates that capture efficiency and sequencing depth are the major biases (Figure 2).

```
> libSize <- dge$samples$lib.size
> plot(libSize/1e3, sizeFactors(sce), log="xy", pch=16, cex=0.7,
+      xlab="Library size (thousands)", ylab="Size factor")
```

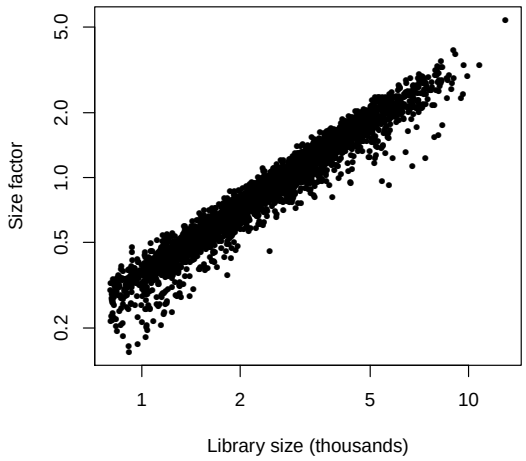

Figure 2: Scatter plot of the estimated size factors against the library size
A strong linear relationship can be seen from the plot.

The normalized log-expresion values are computed as follows.

```
> sce <- normalize(sce)
```

3.3 Modelling the mean-variance trend

The *scran* package works best with spike-ins. In general, *scran* requires spike-ins to compute the technical variation. It then decomposes the total variation into the technical and the biological components for each gene. This is to remove the unwanted technical noise from the data and to only focus on the biological variation.

However, most 10X data don't have spike-in transcripts. One option is to estimate the variances for all genes, fit a trend to the variances and use this trend for computing the technical components of all the genes. (Note this approach is somewhat conservative as the estimated technical components are usually bigger than they actual are.)

```
> fit <- trendVar(sce, use.spikes=FALSE, loess.args=list(span=0.05))
```

We decompose the variance for each gene using the fitted trend, and examine the genes with the highest biological components.

```
> dec <- decomposeVar(fit=fit)
> top.dec <- dec[order(dec$bio, decreasing=TRUE), ]
> top.dec <- as.data.frame(top.dec)
> head(top.dec, n=10L)
```

```
      mean total  bio  tech p.value FDR
Trf   2.26  7.80 6.74 1.064       0   0
Acta2 2.29  7.33 6.26 1.062       0   0
Spp1  1.33  5.95 4.98 0.971       0   0
Apoe  2.39  5.71 4.65 1.057       0   0
Ptn   2.47  5.65 4.59 1.052       0   0
Krt14 1.83  5.33 4.26 1.072       0   0
Krt18 3.56  4.96 4.01 0.943       0   0
Krt19 2.60  5.04 3.99 1.041       0   0
Gpx3  1.63  4.78 3.73 1.050       0   0
Krt17 1.34  3.92 2.94 0.976       0   0
```

We select the top 1000 highly variable genes (HVGs) for the downstream analysis. We produce a mean-variance plot for all the genes and highlight the selected top 1000 HVGs (Figure 3).

```
> top <- 1000
> hvg <- rownames(top.dec)[1:top]
> plot(fit$mean, fit$var, pch=16, cex=0.7)
> curve(fit$trend(x), col="blue", add=TRUE, lwd=2)
> points(fit$mean[hvg], fit$var[hvg], col="red", pch=16, cex=0.7)
```

We can plot the distribution of the expression profile for genes with the largest biological components, to verify that they are indeed highly variable (Figure 4).

```
> plotExpression(sce, features=rownames(top.dec)[1:10])
```

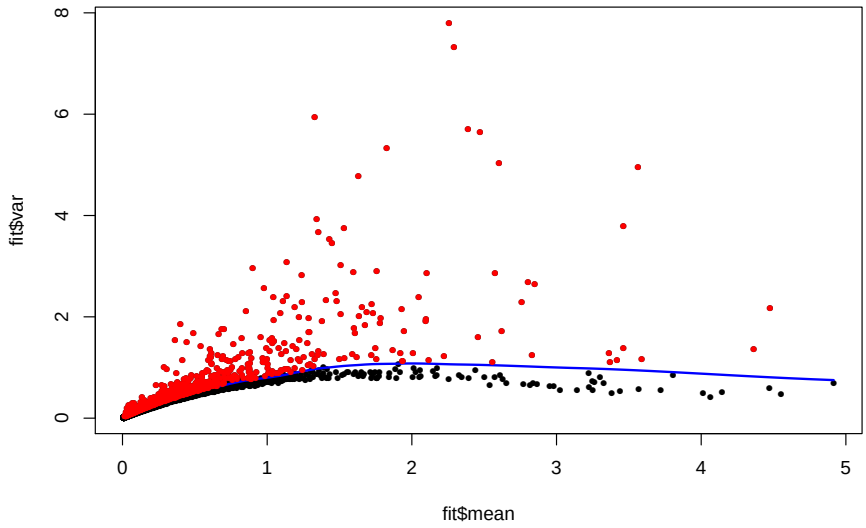

Figure 3: Mean-variance relationship of the 10X data
Each dot represents a gene. The blue line represents the fitted mean-variance trend. The top highly variable genes are highlighted in red.

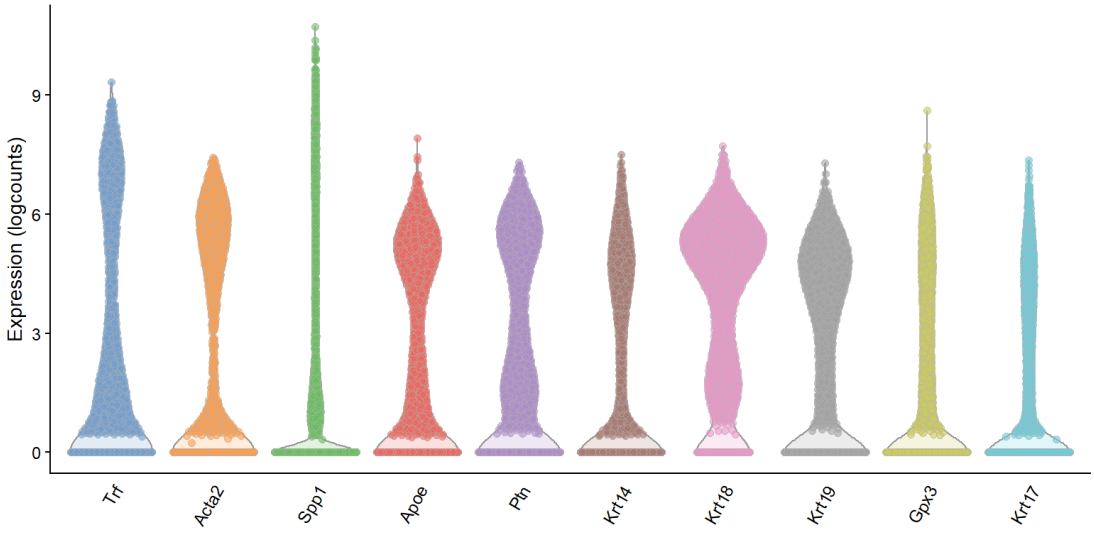

Figure 4: Violin plots of the top 10 most variable genes
The normalized log-expression values of the top genes are indeed highly variable across the cells.

3.4 Principal component analysis

We use the `denoisePCA()` function to compute the principal components (PCs) and choose the number of dimensions to retain. The range of PC dimensions is set to be between 10 and 30.

```
> sce <- denoisePCA(sce, technical=fit$trend, subset.row=hvg,
+     min.rank=10, max.rank=30, approx=TRUE)
> ncol(reducedDim(sce, "PCA"))
```

```
[1] 10
```

We produce pair-wise scatter plots for the first three PCs (Figure 5). Some strong substructure can be seen from the plots.

```
> colData(sce) <- cbind(colData(sce), log10LibSize=log10(libSize))
> plotPCA(sce, ncomponents=3, colour_by="log10LibSize", add_ticks=FALSE)
```

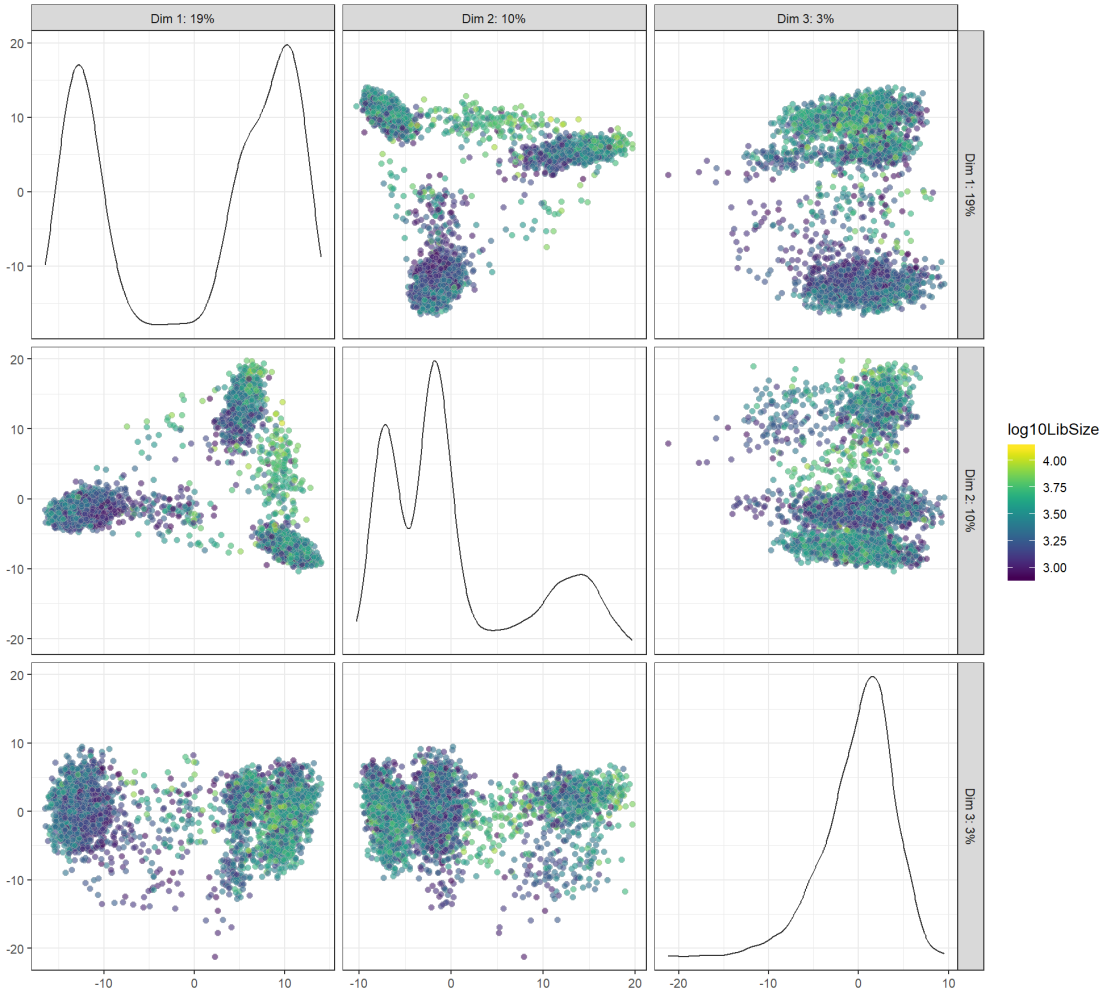

Figure 5: Pairwise PCA plots of the first three PCs
Cells are separated into several clusters in the first three dimensions.

3.5 t-SNE visualization

A widely used approach for dimensionality reduction is the t-stochastic neighbour embedding (t-SNE) method [3]. It is particularly designed for embedding high-dimensional data into a space of two or three dimensions, which can then be visualized in a scatter plot. However, the t-SNE algorithm is computationally expensive and requires the user to consider parameters such as the random seed and perplexity. The perplexity value controls how many nearest neighbours (the size of a cluster) are taken into account when constructing the embedding in the low-dimensional space. A random seed is chosen for getting reproducible results.

```
> set.seed(100)
> sce <- runTSNE(sce, use_dimred="PCA", perplexity=100, theta=0)
```

We create a t-SNE plot and colour the cells by library size (Figure 6).

```
> plotTSNE(sce, colour_by="log10LibSize", add_ticks=FALSE)
```

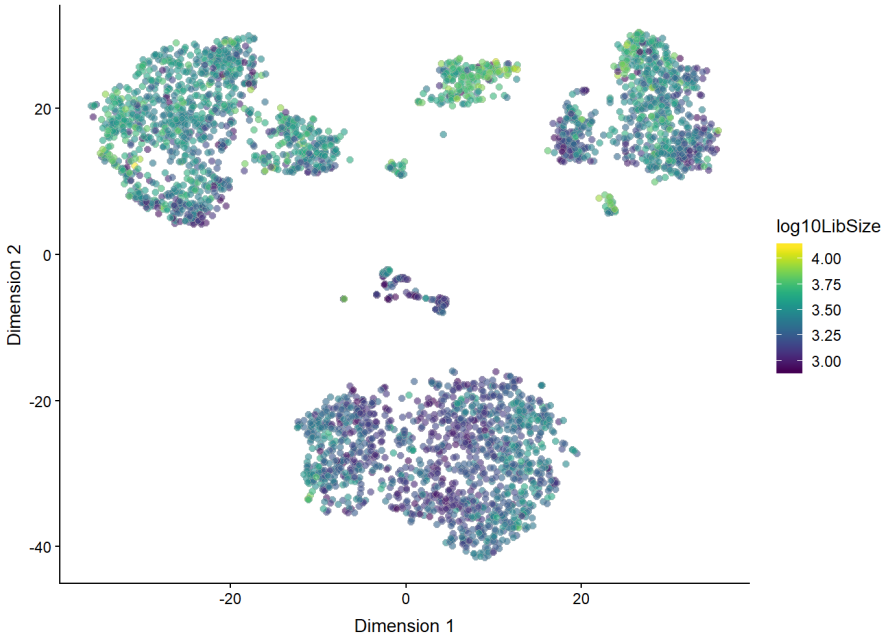

Figure 6: t-SNE visualization of the 10X data
Cells are coloured by library size.

We can also view the expression level of particular genes across all the cells in t-SNE plots. Here, cells are coloured by the expression levels of some epithelial markers such as Krt14 and Krt18 (Figure 7).

```
> p1 <- plotTSNE(sce, colour_by="Krt14", add_ticks=FALSE) + ggtitle("Krt14")
> p2 <- plotTSNE(sce, colour_by="Krt18", add_ticks=FALSE) + ggtitle("Krt18")
> multiplot(p1, p2, cols=2)
```

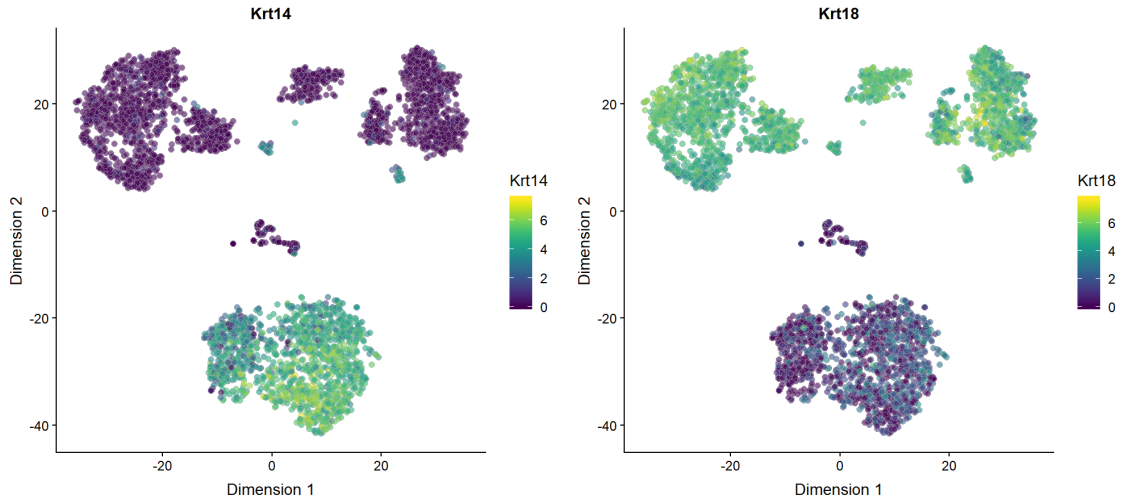

Figure 7: t-SNE visualization with gene expression information
Cells are coloured by the expression levels of genes of interest.

3.6 Cell clustering

One of the main purposes of conducting single-cell experiment is to study the composition of cell populations. One option to perform cell clustering in *scran* is to build a shared nearest neighbour (SNN) graph [4]. It then uses the Walktrap algorithm to identify clusters.

```
> snn_k60 <- buildSNNGraph(sce, k=60, use.dimred="PCA")
> wt_k60 <- igraph::cluster_walktrap(snn_k60)
> clst_k60 <- factor(wt_k60$membership)
> table(clst_k60)

clst_k60
   1    2    3    4    5    6
1085  696  121  901  141  358
```

The parameter `k` of `buildSNNGraph` determines the number of neighbours of each cell. Increasing `k` reduces the number of clusters, and vice versa.

```
> snn_k70 <- buildSNNGraph(sce, k=70, use.dimred="PCA")
> wt_k70 <- igraph::cluster_walktrap(snn_k70)
> clst_k70 <- factor(wt_k70$membership)
> table(clst_k70)

clst_k70
   1    2    3    4
 821 1259 1101  121
```

We examine the cluster identities on a t-SNE plot to confirm that different clusters are indeed separated (Figure 8).

```
> sce$Cluster <- clst_k60
> k60 <- plotTSNE(sce, colour_by="Cluster", add_ticks=FALSE) +
+     ggtitle("SNN clutering with k=60")
```

```
> sce$Cluster <- clst_k70
> k70 <- plotTSNE(sce, colour_by="Cluster", add_ticks=FALSE) +
+     ggtitle("SNN clutering with k=70")
> multiplot(k60, k70, cols=2)
```

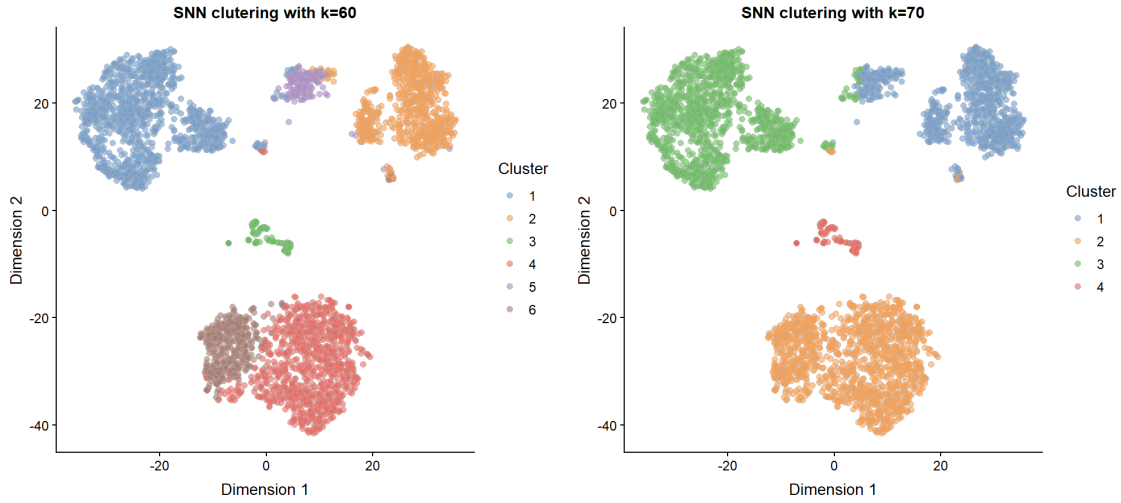

Figure 8: t-SNE visualization with cell clustering information
Cells are coloured by cluster defined using SNN with k=60 and k=70, respectively.

If a satisfactory clustering can not obtained by SNN, we can try other approaches such as hierarchical clustering (Figure 9).

```
> hclst <- quickCluster(sce, subset.row=hvg, assay.type="logcounts",
+     method="hclust", min.size=50, min.mean=0.5)
> table(hclst)

hclst
   1    2    3    4    5
1250 1079  682  171  120

> sce$Cluster <- factor(hclst)
> plotTSNE(sce, colour_by="Cluster", add_ticks=FALSE) +
+     ggtitle("Hierarchical clustering")
```

3.7 Marker gene detection

We perform DE analysis to detect marker genes for each cluster. Here, we are only interested in upregulated genes in each cluster, as these are more useful for positive identification of cell types in a heterogeneous population.

```
> markers <- findMarkers(sce, clusters=sce$Cluster, direction="up")
```

The top markers of the first cluster are shown as follows.

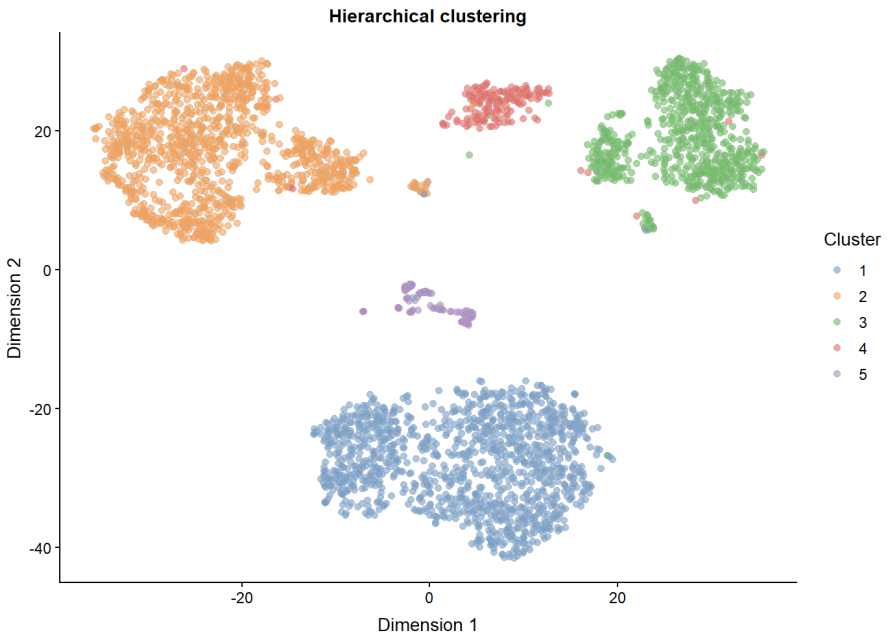

Figure 9: **t-SNE visualization with cell clustering information**
Cells are coloured by cluster defined using hierarchical clustering.

```
> marker.set <- as.data.frame(markers[["1"]])
> head(marker.set, n=20L)
```

	Top	p.value	FDR	logFC.2	logFC.3	logFC.4	logFC.5
Apoe	1	0.00e+00	0.00e+00	4.901	3.864	4.355	3.348
Acta2	1	0.00e+00	0.00e+00	5.288	5.244	5.270	3.568
Krt14	1	0.00e+00	0.00e+00	4.378	4.338	4.439	4.365
1500015O10Rik	1	0.00e+00	0.00e+00	1.941	1.906	1.944	1.968
Cxcl14	2	0.00e+00	0.00e+00	3.568	3.519	3.571	3.561
Myl9	3	0.00e+00	0.00e+00	2.677	2.651	2.698	0.995
Krt5	3	4.39e-245	1.36e-242	1.284	1.281	1.300	1.299
Tagln	4	0.00e+00	0.00e+00	3.654	3.561	3.667	2.173
Rbp1	4	2.72e-194	5.57e-192	0.952	0.810	0.841	0.969
Tpm2	5	0.00e+00	0.00e+00	3.420	3.384	3.440	1.775
Cnn1	5	0.00e+00	0.00e+00	3.052	3.010	3.027	2.819
Wif1	5	5.54e-180	9.69e-178	1.084	1.064	1.081	1.097
Epcam	6	1.95e-179	3.35e-177	-2.130	-2.460	-2.324	1.436
Sparc	7	0.00e+00	0.00e+00	2.333	2.336	2.344	-0.377
Krt17	7	0.00e+00	0.00e+00	3.209	3.191	3.252	3.234
Csrp1	8	0.00e+00	0.00e+00	2.385	1.895	2.100	1.569
Sfn	8	1.87e-310	7.82e-308	2.257	1.090	1.292	3.928
Ifitm3	9	0.00e+00	0.00e+00	1.575	2.187	1.805	0.762
Tfap2c	9	4.21e-161	6.04e-159	0.855	0.617	0.692	0.902
Lcn2	10	0.00e+00	0.00e+00	2.024	-1.767	-0.688	2.025

We select the top 10 markers of all the clusters and then create a heat map (Figure 10).

```
> top10 <- lapply(markers, "[", 1:10, )
> GSet <- as.vector(sapply(top10, "rownames"))
> GSet <- GSet[!duplicated(GSet)]
> GSet
 [1] "Apoe"          "Acta2"         "Krt14"         "1500015O10Rik" "Cxcl14"
 [6] "Myl9"          "Krt5"          "Tagln"         "Rbp1"          "Tpm2"
[11] "Krt19"         "Ptn"           "Krt7"          "Wfdc2"         "Stc2"
[16] "Krt18"         "Prlr"          "Ly6d"          "Gpx3"          "Epcam"
[21] "Trf"           "Mfge8"         "Mgst1"         "Lcn2"          "Cd81"
[26] "Plet1"         "Itm2b"         "Csn3"          "Krt8"          "Vim"
[31] "Sparc"         "Id3"           "Fth1"          "Igfbp7"        "Rarres2"
[36] "Gsn"           "Tuba1a"        "Serpinh1"
```

```
> library(pheatmap)
> plotHeatmap(sce, features=GSet, exprs_values="logcounts",
+     color=colorRampPalette(c("blue","white","red"))(100),
+     columns=order(sce$Cluster), labels_col="",
+     colour_columns_by="Cluster", cluster_cols=FALSE, center=TRUE,
+     symmetric=TRUE, zlim=c(-3, 3))
```

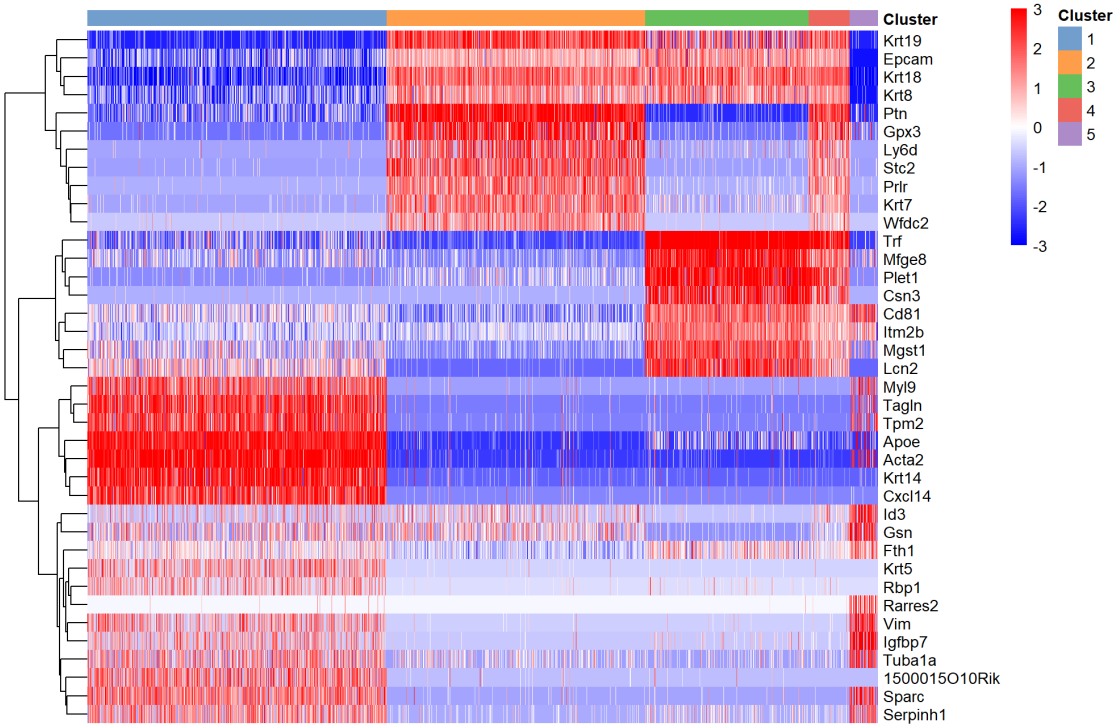

Figure 10: Heat map of the top 10 markers of all clusters
Expression patterns of marker genes can be seen from the heat map.

4 Correlating with bulk RNA-seq profiles

4.1 Gene signatures from a previous study

The t-SNE plot and cell clustering algorithm grouped the cells into three major clusters and two minor clusters. To reveal the identities of each cluster, we use prior knowledge from a previous bulk RNA-seq experiment [5]. In particular, we use the signature genes of the three major epithelial populations: basal, LP and ML, obtained from a DE analysis of a RNA-seq data. The gene signatures are available as an `RData` file at http://bioinf.wehi.edu.au/edgeR/BulkSignatures.RData.

We first load in the gene signatures of the three populations.

```
> load("Data/BulkSignatures.RData")
> head(Basal)

[1] "Xkr4"          "Cpa6"          "A830018L16Rik" "Sulf1"         "Eya1"
[6] "Sbspon"

> head(LP)

[1] "Ogfrl1" "Ptpn18" "Mgat4a" "Npas2"  "Slc9a4" "Slc9a2"

> head(ML)

[1] "Msc"     "Plekhb2" "Nck2"    "Dnpep"   "B3gnt7"  "Mlph"
```

We restrict the gene signatures to those expressed (not filtered) in the 10X data.

```
> Basal <- intersect(Basal, dge$genes$Official)
> LP <- intersect(LP, dge$genes$Official)
> ML <- intersect(ML, dge$genes$Official)
```

4.2 Ternary plot visualization

We produce a ternary plot to see which of the three populations the cells are closer to. To measure the similarity between each cell and three populations, we count the numbers of expressed signatures (with at least 1 count) in each cell. The position of each cell on the ternary plot is determined by the numbers of expressed gene signatures of the three populations in that cell.

```
> TN <- matrix(0L, ncol(dge), 3L)
> colnames(TN) <- c("Basal", "LP", "ML")
> TN[, "Basal"] <- colSums(dge$counts[Basal,] > 0L)
> TN[, "LP"] <- colSums(dge$counts[LP,] > 0L)
> TN[, "ML"] <- colSums(dge$counts[ML,] > 0L)
> dim(TN)

[1] 3302    3

> head(TN)

     Basal LP  ML
[1,]    23 25 102
```

```
[2,]    69 10   7
[3,]    22 67  16
[4,]    18 12  85
[5,]     7 14  55
[6,]    29 24 122
```

A ternary plot is produced using the `ternaryplot` function of the *vcd* package (Figure 11). We use the *scater* colour palette to make sure that the colours in the ternary plot are consistent to those in the t-SNE plot.

```
> ncls <- nlevels(sce$Cluster)
> col.p <- scater:::.get_palette("tableau10medium")
> library(vcd)
> ternaryplot(TN[,c(2,3,1)], cex=0.25, pch=16, col=col.p[sce$Cluster],
+     main="Mouse bulk RNA-seq signatures", grid=TRUE)
> grid_legend(0.8, 0.7, pch=16, col=col.p[1:ncls],
+     labels=paste0("Cluster", 1:ncls, " - ", table(sce$Cluster)),
+     title="Cluster", frame=FALSE, size=0.5)
```

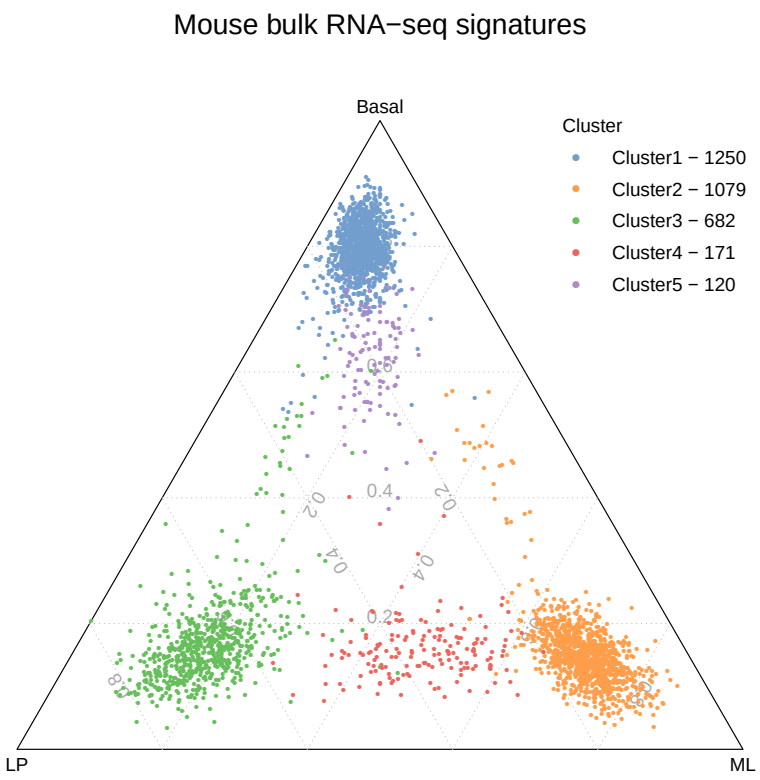

Figure 11: Ternary plot
Cells are plotted according to the relative proportion of genes they express that are specific to the three cell types.

Total single epithelial cells are classified into basal, LP, and ML cell clusters based on known lineage-specific signatures for these cell types. This analysis also shows the presence of the luminal intermediate cluster between LP and ML cells.

4.3 Signature scores

Ternary plots are useful for visualizing similarities between different datasets. However, ternary plots only work with 3 distinct group conditions. A more general approach of correlating datasets with prior knowledge from other studies is to calculate signature scores using gene signatures.

To do this, we first compute the log-CPM values of all the genes across all the cells. Then we subset the matrix of the log-CPM values by the gene signatures of each cell type.

```
> lcpm <- edgeR::cpm(dge, log=TRUE, prior.count=3)
> lcpm_Basal <- lcpm[Basal, ]
> lcpm_LP <- lcpm[LP, ]
> lcpm_ML <- lcpm[ML, ]
```

For a particular cell, the signature score of a given cell type is defined as the average log-CPM value of all the signatures of that cell type. We can easily calculate the signature scores for all the cells in all three cell types as follows.

```
> score_Basal <- colMeans(lcpm_Basal)
> score_LP <- colMeans(lcpm_LP)
> score_ML <- colMeans(lcpm_ML)
```

A cluster of cells with high expression of gene signatures of a particular cell type would have high signature scores of that cell type. Boxplots can be produced to visualize the results (Figure 12).

```
> par(mfrow=c(2,2))
> boxplot(score_Basal ~ sce$Cluster, xlab="Cluster", ylab="AveLogCPM",
+     varwidth=TRUE, col=col.p[1:ncls], main="Basal signature score")
> boxplot(score_LP ~ sce$Cluster, xlab="Cluster", ylab="AveLogCPM",
+     varwidth=TRUE, col=col.p[1:ncls], main="LP signature score")
> boxplot(score_ML ~ sce$Cluster, xlab="Cluster", ylab="AveLogCPM",
+     varwidth=TRUE, col=col.p[1:ncls], main="ML signature score")
```

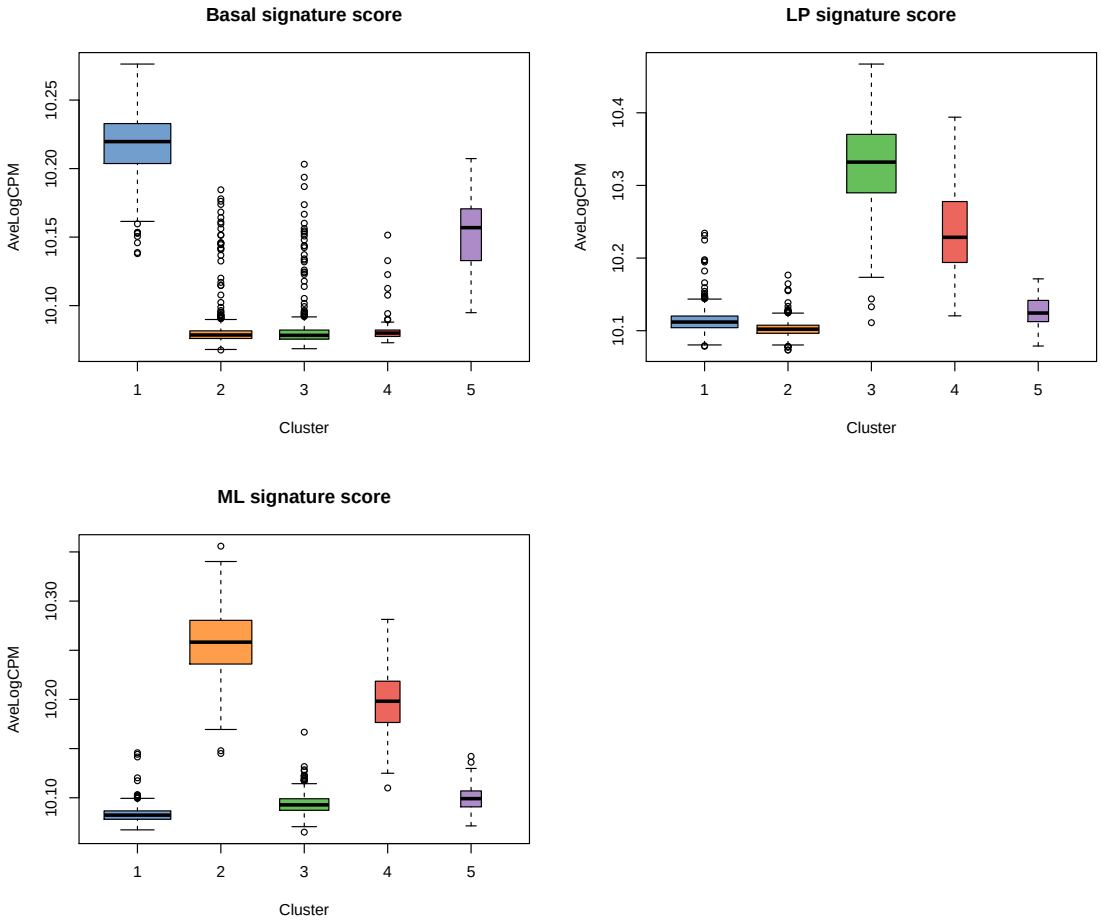

Figure 12: **Boxplots of signature scores for different cell types**
Cells from the same cluster are grouped together in one box.

5 Diffusion analysis

5.1 Create a `CellDataSet` object

Diffusion maps can be useful for dealing with the problem of defining differentiation trajectories. Here, we use the *monocle* package which adopts a strategy of ordering single cells in pseudotime.

monocle holds single cell expression data in objects of the `CellDataSet` class. We can easily convert a `SingleCellExperiment` object to a `CellDataSet` object as follows.

```
> library(monocle)
> rowData(sce) <- data.frame(gene_short_name=rownames(sce))
> cds <- convertTo(sce, type="monocle", row.fields=1)
> pData(cds)$Cluster <- sce$Cluster
```

5.2 Dispersion estimation

Inferring a single-cell trajectory is a machine learning problem. The first step is to select the genes *monocle* will use as input for its machine learning approach. Here, we use the highly variable genes (HVGs) defined previously to order the cells.

We first estimate the dispersion values of each gene across all the cells.

```
> cds <- estimateSizeFactors(cds)
> cds <- estimateDispersions(cds)
```

We highlight the HVGs on the dispersion plot (Figure 13). This is to confirm that the selected HVGs have indeed large dispersions.

```
> cds <- setOrderingFilter(cds, ordering_genes=hvg)
> plot_ordering_genes(cds)
```

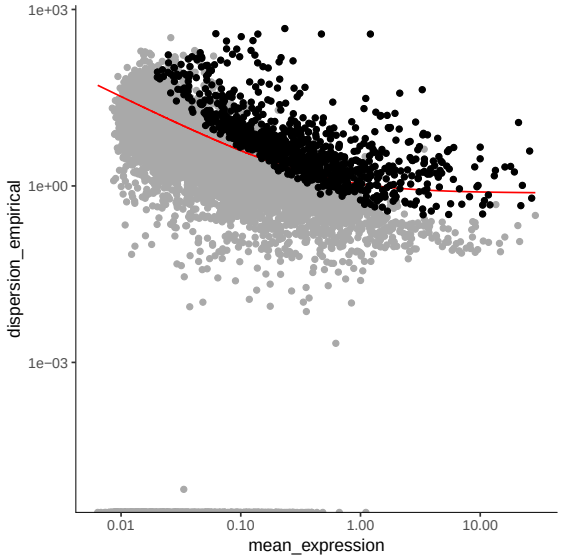

Figure 13: Mean-dispersion plot by monocle
Highly variable genes are highlighted in black.

5.3 Ordering cells in pseudotime

Once genes are selected for ordering cells, *monocle* applies a dimensionality reduction to the data. In a lower dimensional space, *monocle* forms the trajectory that describes how cells transition from one state into another. It assumes that the trajectory has a tree structure and fits the best tree it can to the data.

```
> cds <- reduceDimension(cds, max_components=2, method='DDRTree')
> cds <- orderCells(cds, reverse=TRUE)
```

The cell ordering and transition can be visualized in a minimum spanning tree (Figure 14).

```
> plot_cell_trajectory(cds, color_by="Pseudotime")
```

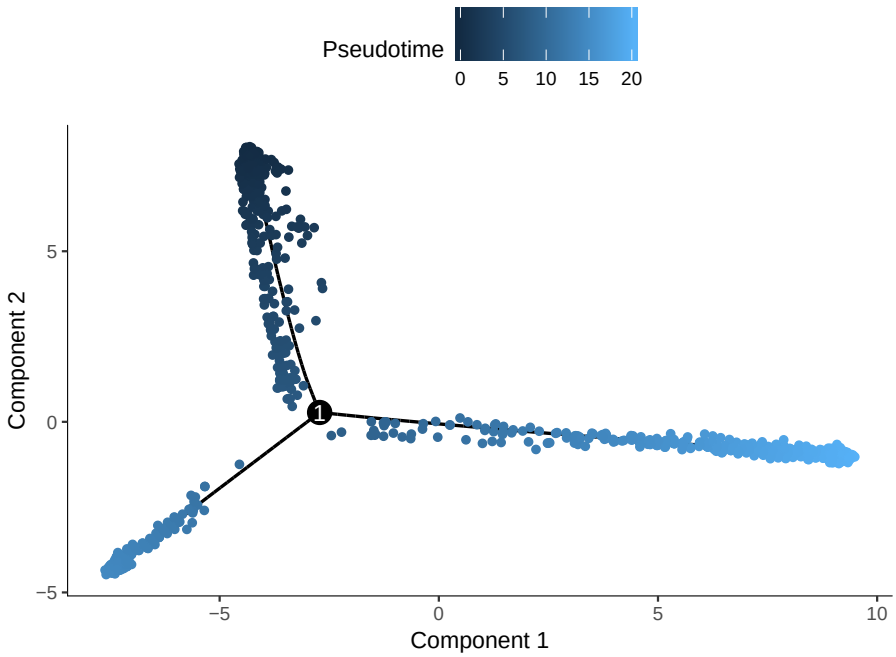

Figure 14: Trajectory plot by monocle
Cells are coloured by pseudo-time.

We also colour the cells by clusters (Figure 15).

```
> g <- plot_cell_trajectory(cds, color_by="Cluster")
> colours <- scater:::.get_palette("tableau10medium")
> g <- g + scale_color_manual(values=colours)
> g
```

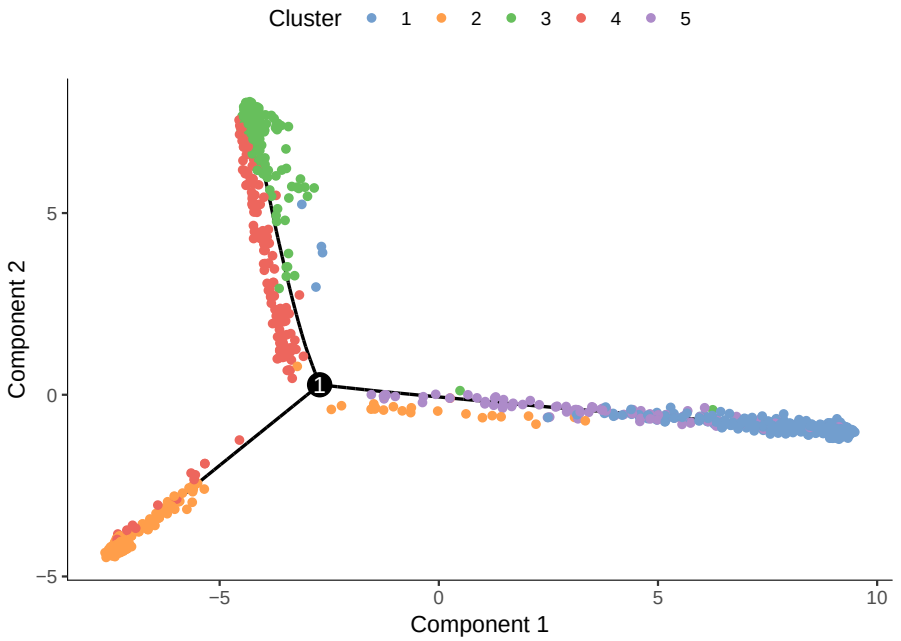

Figure 15: Trajectory plot by monocle
Cells are coloured by clusters defined previously.

6 Data and software availability

All data analyzed in the workflow is available for download from public websites. All software used is publicly available as part of Bioconductor and the Comprehensive R Archive Network (CRAN). The article includes the complete code necessary to reproduce the analyses shown.

This workflow depends on various packages from version 3.8 of the Bioconductor project, running on R version 3.5.1. The complete list of the packages used for this workflow are shown below:

```
> sessionInfo()

R version 3.5.1 (2018-07-02)
Platform: x86_64-w64-mingw32/x64 (64-bit)
Running under: Windows 10 x64 (build 16299)

Matrix products: default

locale:
[1] LC_COLLATE=English_Australia.1252  LC_CTYPE=English_Australia.1252
[3] LC_MONETARY=English_Australia.1252 LC_NUMERIC=C
[5] LC_TIME=English_Australia.1252

attached base packages:
 [1] splines   grid      parallel  stats4    stats     graphics  grDevices utils
 [9] datasets  methods   base
```

```
other attached packages:
 [1] bindrcpp_0.2.2             monocle_2.10.0             DDRTree_0.1.5
 [4] irlba_2.3.2                VGAM_1.0-6                 Matrix_1.2-15
 [7] vcd_1.4-4                  pheatmap_1.0.10            edgeR_3.24.0
[10] limma_3.38.2               scran_1.10.1               scater_1.10.0
[13] ggplot2_3.1.0              SingleCellExperiment_1.4.0 SummarizedExperiment_1.12.0
[16] DelayedArray_0.8.0         BiocParallel_1.16.0        matrixStats_0.54.0
[19] Biobase_2.42.0             GenomicRanges_1.34.0       GenomeInfoDb_1.18.1
[22] IRanges_2.16.0             S4Vectors_0.20.1           BiocGenerics_0.28.0
[25] knitr_1.20

loaded via a namespace (and not attached):
 [1] bitops_1.0-6             RColorBrewer_1.1-2       rprojroot_1.3-2
 [4] docopt_0.6.1             dynamicTreeCut_1.63-1    tools_3.5.1
 [7] backports_1.1.2          R6_2.3.0                 HDF5Array_1.10.0
[10] vipor_0.4.5              lazyeval_0.2.1           colorspace_1.3-2
[13] withr_2.1.2              tidyselect_0.2.5         gridExtra_2.3
[16] compiler_3.5.1           BiocNeighbors_1.0.0      slam_0.1-43
[19] labeling_0.3             scales_1.0.0             lmtest_0.9-36
[22] proxy_0.4-22             stringr_1.3.1            digest_0.6.18
[25] sparsesvd_0.1-4          rmarkdown_1.10           XVector_0.22.0
[28] pkgconfig_2.0.2          htmltools_0.3.6          highr_0.7
[31] rlang_0.3.0.1            FNN_1.1.2.1              DelayedMatrixStats_1.4.0
[34] bindr_0.1.1              combinat_0.0-8           zoo_1.8-4
[37] dplyr_0.7.8              RCurl_1.95-4.11          magrittr_1.5
[40] GenomeInfoDbData_1.2.0   Rcpp_1.0.0               ggbeeswarm_0.6.0
[43] munsell_0.5.0            Rhdf5lib_1.4.0           viridis_0.5.1
[46] stringi_1.2.4            yaml_2.2.0               MASS_7.3-51.1
[49] zlibbioc_1.28.0          rhdf5_2.26.0             Rtsne_0.15
[52] plyr_1.8.4               ggrepel_0.8.0            crayon_1.3.4
[55] lattice_0.20-38          cowplot_0.9.3            locfit_1.5-9.1
[58] pillar_1.3.0             igraph_1.2.2             reshape2_1.4.3
[61] codetools_0.2-15         glue_1.3.0               evaluate_0.12
[64] BiocManager_1.30.4       RANN_2.6                 gtable_0.2.0
[67] purrr_0.2.5              assertthat_0.2.0         qlcMatrix_0.9.7
[70] HSMMSingleCell_1.2.0     viridisLite_0.3.0        tibble_1.4.2
[73] beeswarm_0.2.3           fastICA_1.2-1            densityClust_0.3
[76] cluster_2.0.7-1          statmod_1.4.30           BiocStyle_2.10.0
```

References

[1] Bhupinder Pal, Yunshun Chen, François Vaillant, Paul Jamieson, Lavinia Gordon, Anne C Rios, Stephen Wilcox, Naiyang Fu, Kevin He Liu, Felicity C Jackling, et al. Construction of developmental lineage relationships in the mouse mammary gland by single-cell rna profiling. *Nature communications*, 8(1):1627, 2017.

[2] Aaron TL Lun, Karsten Bach, and John C Marioni. Pooling across cells to normalize single-cell rna sequencing data with many zero counts. *Genome biology*, 17(1):75, 2016.

[3] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(Nov):2579–2605, 2008.

[4] Chen Xu and Zhengchang Su. Identification of cell types from single-cell transcriptomes using a novel clustering method. *Bioinformatics*, 31(12):1974–1980, 2015.

[5] Julie M Sheridan, Matthew E Ritchie, Sarah A Best, Kun Jiang, Tamara J Beck, François Vaillant, Kevin Liu, Ross A Dickins, Gordon K Smyth, Geoffrey J Lindeman, et al. A pooled shrna screen for regulators of primary mammary stem and progenitor cells identifies roles for asap1 and prox1. *BMC cancer*, 15(1):221, 2015.