Detecting doublets in single-cell RNA-seq data

Aaron T. L. Lun[1]

[1]Cancer Research UK Cambridge Institute, Li Ka Shing Centre, Robinson Way, Cambridge CB2 0RE, United Kingdom

2019-01-09

1 Overview

In single-cell RNA sequencing (scRNA-seq) experiments, doublets are artifactual libraries generated from two cells. They typically arise due to errors in cell sorting or capture, especially in droplet-based protocols (Zheng et al. 2017) involving thousands of cells. Doublets are obviously undesirable when the aim is to characterize populations at the *single*-cell level. In particular, they can incorrectly suggest the existence of intermediate populations or transitory states that not actually exist. Thus, it is desirable to remove doublet libraries so that they do not compromise interpretation of the results.

Several experimental strategies are available for doublet removal. One approach exploits natural genetic variation when pooling cells from multiple donor individuals (Kang et al. 2018). Doublets can be identified as libraries with allele combinations that do not exist in any single donor. Another approach is to mark a subset of cells (e.g., all cells from one sample) with an antibody conjugated to a different oligonucleotide (Stoeckius et al. 2017). Upon pooling, libraries that are observed to have different oligonucleotides are considered to be doublets and removed.

These approaches can be highly effective but rely on experimental information that may not be available.

A more general approach is to infer doublets from the expression profiles alone (Dahlin et al. 2018). In this workflow, we will describe two purely computational approaches for detecting doublets from scRNA-seq data. The main difference between These two methods is whether or not they need cluster information beforehand. Both are implemented in the *scran (https://bioconductor.org/packages/3.8/scran)* package from the open-source Bioconductor project (Huber et al. 2015). We will demonstrate the use of these methods on data from a droplet-based scRNA-seq study of the mouse mammary gland (Bach et al. 2017), available from NCBI GEO with the accession code GSE106273.

```
library(BiocFileCache)
bfc <- BiocFileCache("raw_data", ask = FALSE)
base.path <- "ftp://ftp.ncbi.nlm.nih.gov/geo/samples/GSM2834
nnn/GSM2834500/suppl"
barcode.fname <- bfcrpath(bfc, file.path(base.path,
    "GSM2834500%5FG%5F1%5Fbarcodes%2Etsv%2Egz"))
gene.fname <- bfcrpath(bfc, file.path(base.path,
    "GSM2834500%5FG%5F1%5Fgenes%2Etsv%2Egz"))
counts.fname <- bfcrpath(bfc, file.path(base.path,
    "GSM2834500%5FG%5F1%5Fmatrix%2Emtx%2Egz"))
```

2 Preparing the data

2.1 Reading in the counts

We create a `SingleCellExperiment` object from the count matrix. The files have been modified from the *CellRanger* output, so we have to manually load them in rather than using `read10xCounts()`.

```
library(scater)
library(Matrix)
gene.info <- read.table(gene.fname, stringsAsFactors=FALSE)
colnames(gene.info) <- c("Ensembl", "Symbol")
sce <- SingleCellExperiment(
    list(counts=as(readMM(counts.fname), "dgCMatrix")),
    rowData=gene.info,
    colData=DataFrame(Barcode=readLines(barcode.fname))
)
```

We put some more meaningful information in the row and column names. Note the use of `uniquifyFeatureNames()` to generate unique row names from gene symbols.

```
rownames(sce) <- uniquifyFeatureNames(
    rowData(sce)$Ensembl, rowData(sce)$Symbol)
colnames(sce) <- sce$Barcode
sce
```

```
## class: SingleCellExperiment
## dim: 27998 2915
## metadata(0):
## assays(1): counts
## rownames(27998): Xkr4 Gm1992 ... Vmn2r122 CAAA01147332.1
## rowData names(2): Ensembl Symbol
## colnames(2915): AAACCTGAGGATGCGT-1 AAACCTGGTAGTAGTA-1 ...
##   TTTGTCATCCTTAATC-1 TTTGTCATCGAACGGA-1
## colData names(1): Barcode
## reducedDimNames(0):
## spikeNames(0):
```

We add some genomic location annotation for downstream use.

```
library(TxDb.Mmusculus.UCSC.mm10.ensGene)
chrloc <- mapIds(TxDb.Mmusculus.UCSC.mm10.ensGene, keytype="
GENEID",
    keys=rowData(sce)$Ensembl, column="CDSCHROM")
rowData(sce)$Chr <- chrloc
```

2.2 Quality control

We compute quality control (QC) metrics using the `calculateQCMetrics()` function from the *scater (https://bioconductor.org/packages/3.8/scater)* package (McCarthy et al. 2017).

```
is.mito <- rowData(sce)$Chr == "chrM"
summary(is.mito)
```

```
##    Mode   FALSE    TRUE    NA's
## logical   21767      13    6218
```

```
sce <- calculateQCMetrics(sce, feature_controls=list(Mito=wh
ich(is.mito)))
```

We remove cells that are outliers for any of these metrics, as previously discussed. Note that some quality control was already performed by the authors of the original study, so relatively few cells are discarded here.

```
low.lib <- isOutlier(sce$total_counts, log=TRUE, nmads=3, ty
pe="lower")
low.nexprs <- isOutlier(sce$total_features_by_counts, log=TR
UE, nmads=3, type="lower")
high.mito <- isOutlier(sce$pct_counts_Mito, nmads=3, type="h
igher")
discard <- low.lib | low.nexprs | high.mito
DataFrame(LowLib=sum(low.lib), LowNum=sum(low.nexprs), HighM
ito=sum(high.mito),
    Discard=sum(discard), Kept=sum(!discard))
```

```
## DataFrame with 1 row and 5 columns
##      LowLib    LowNum  HighMito   Discard      Kept
##   <integer> <integer> <integer> <integer> <integer>
## 1         0         0       143       143      2772
```

We then subset the `SingleCellExperiment` object to remove these low-quality cells.

```
sce <- sce[,!discard]
```

2.3 Normalization for cell-specific biases

We apply the deconvolution method with pre-clustering (Lun, Bach, and Marioni 2016) to compute size factors for scaling normalization of cell-specific biases.

```
library(scran)
set.seed(1000)
clusters <- quickCluster(sce, method="igraph", min.mean=0.1)
table(clusters)
```

```
## clusters
##    1    2    3
##  919 1045  808
```

```
sce <- computeSumFactors(sce, clusters=clusters, min.mean=0.
1)
summary(sizeFactors(sce))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2688  0.5310  0.7672  1.0000  1.1987 10.6856
```

We then compute log-normalized expression values for downstream use. This data set does not contain spike-in transcripts so separate normalziation with `computeSpikeFactors()` is not required.

```
sce <- normalize(sce)
assayNames(sce)
```

```
## [1] "counts"    "logcounts"
```

2.4 Modelling and removing noise

As we have no spike-ins, we model technical noise using the `makeTechTrend()` function.

```
tech.trend <- makeTechTrend(x=sce)
fit <- trendVar(sce, use.spikes=FALSE)
plot(fit$mean, fit$var, pch=16,
    xlab="Mean log-expression",
    ylab="Variance of log-expression")
curve(tech.trend(x), add=TRUE, col="red")
```

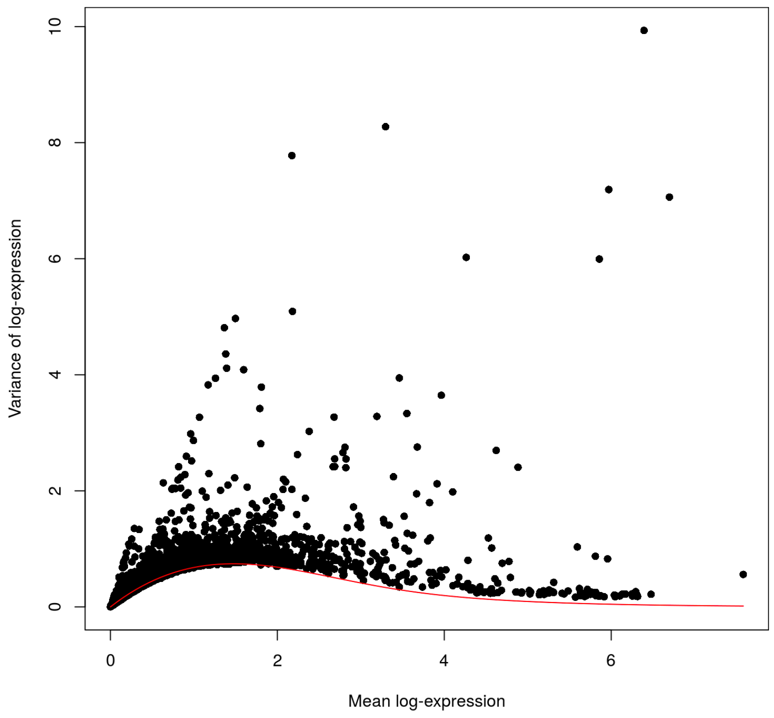


Figure 1: Variance of the log-expression values as a function of the mean log-expression in the mammary gland data set. Each point represents a gene, and the red line corresponds to Poisson variance.

We use `denoisePCA()` to choose the number of principal components (PCs) to retain based on the technical noise per gene. We need to set the seed for reproducibility when `approximate=TRUE`, due to the use of randomized methods from *irlba (https://bioconductor.org/packages/3.8/irlba)*.

```
set.seed(12345)
sce <- denoisePCA(sce, technical=tech.trend, approximate=TRU
E)
ncol(reducedDim(sce))
```

```
## [1] 14
```

2.5 Clustering into subpopulations

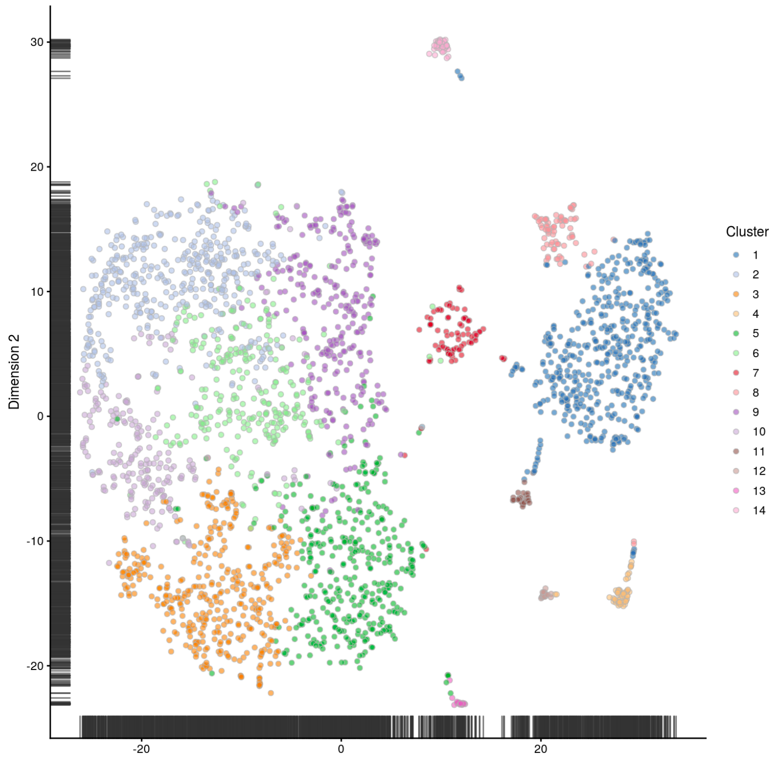
We cluster cells into putative subpopulations using `buildSNNGraph()` (Xu and Su 2015). We use a higher `k` to increase connectivity and reduce the granularity of the clustering.

```
snn.gr <- buildSNNGraph(sce, use.dimred="PCA", k=25)
sce$Cluster <- factor(igraph::cluster_walktrap(snn.gr)$membe
rship)
table(sce$Cluster)
```

```
##
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14
## 475 411 398  54 378 271  84  74 289 219  33  22  25  39
```

We visualize the clustering on a *t*-SNE plot (Van der Maaten and Hinton 2008). Figure 2 shows that there are a number of well-separated clusters as well as some more inter-related clusters.

```
set.seed(1000)
sce <- runTSNE(sce, use_dimred="PCA")
plotTSNE(sce, colour_by="Cluster")
```



Dimension 1

Figure 2: t-SNE plot of the mammary gland data set. Each point is a cell coloured according to its assigned cluster identity.

3 Doublet detection with clusters

The `doubletCluster()` function will identify clusters that have intermediate expression profiles of two other clusters (Bach et al. 2017). Specifically, it will examine every possible triplet of clusters consisting of a query cluster and its two “parents”. It will then compute a number of statistics:

- The number of genes (`N`) that are differentially expressed in the same direction in the query cluster compared to *both* of the parent clusters. Such genes would be unique markers for the query cluster and provide evidence against the null hypothesis, i.e., that the query cluster consists of doublets from the two parents. Clusters with few unique genes are more likely to be doublets.
- The ratio of the median library size in each parent to the median library size in the query (`lib.size*`). Doublet libraries are generated from a larger initial pool of RNA compared to libraries for single cells, and thus the former should have larger library sizes. Library size ratios much greater than unity are inconsistent with a doublet identity for the query.
- The proportion of cells in the query cluster should also be reasonable - typically less than 5% of all cells, depending on how many cells were loaded onto the 10X Genomics device.

```
dbl.out <- doubletCluster(sce, sce$Cluster)
dbl.out
```

```
## DataFrame with 14 rows and 9 columns
##         source1     source2         N        best
p.value
##     <character> <character> <integer> <character>
<numeric>
## 7             8           5         0       Fabp3   0.051
6239051106075
## 10           13           2        13        Xist 1.73868
501814698e-29
## 8            12           1        28         Ptn 1.10432
261800312e-14
## 5            12           7        49       Cotl1 7.90164
177238926e-08
## 6             7           2        56      Sec61b  1.3329
887878903e-07
## ...         ...         ...       ...         ...
...
## 2            12           6       182       Cdc20  1.4775
692594461e-19
## 13           14          12       198        Gpx3  1.1282
711558589e-19
## 4            14          12       270        C1qb 9.41841
774529017e-49
## 11           14          12       299       Fabp4 2.70725
398963721e-32
## 14           13          11       388         Dcn 4.93706
079643116e-32
##             lib.size1         lib.size2                pr
op
##             <numeric>         <numeric>           <numeri
c>
## 7   0.456830005034402 0.574593052525592  0.03030303030303
03
## 10  0.677851605758582  1.51240310077519   0.0790043290043
29
## 8     0.9924694645973  1.18853889246028  0.02669552669552
67
## 5           0.7890625  1.74036214953271   0.1363636363636
36
## 6   0.509882775733721 0.584281680499701  0.09776334776334
78
## ...               ...               ...
...
## 2   0.395657904371385  1.71150325840228   0.1482683982683
98
## 13  0.882698905407613 0.882780591406633 0.009018759018759
02
## 4   0.856192060850963 0.856271293875287  0.01948051948051
95
## 11  0.666050295857988 0.666111932938856  0.01190476190476
19
## 14   1.13288913566537  1.50138811771238  0.01406926406926
41
##                                           all.pairs
##                                    <DataFrameList>
## 7              8:5:0:...,6:1:1:...,9:1:3:...,...
## 10       13:2:13:...,12:2:26:...,12:3:37:...,...
## 8       12:1:28:...,11:1:97:...,13:1:143:...,...
## 5        12:7:49:...,12:9:55:...,13:9:96:...,...
## 6       7:2:56:...,10:9:145:...,10:7:146:...,...
## ...                                          ...
## 2     12:6:182:...,10:9:248:...,10:8:267:...,...
## 13   14:12:198:...,14:10:200:...,12:3:202:...,...
## 4   14:12:270:...,12:11:331:...,13:12:371:...,...
## 11   14:12:299:...,14:13:329:...,12:1:338:...,...
## 14     13:11:388:...,11:4:406:...,8:4:434:...,...
```

Examination of the output of `doubletCluster()` indicates that cluster 7 has the fewest unique genes and library sizes that are comparable to or greater than its parents. We see that every gene detected in this cluster is also expressed in either of the two proposed parent clusters (Figure 3).

```
markers <- findMarkers(sce, sce$Cluster, direction="up")
dbl.markers <- markers[["7"]]
chosen <- rownames(dbl.markers)[dbl.markers$Top <= 10]
plotHeatmap(sce, columns=order(sce$Cluster), colour_columns_
by="Cluster",
    features=chosen, cluster_cols=FALSE, center=TRUE, symmet
ric=TRUE,
    zlim=c(-5, 5), show_colnames=FALSE)
```

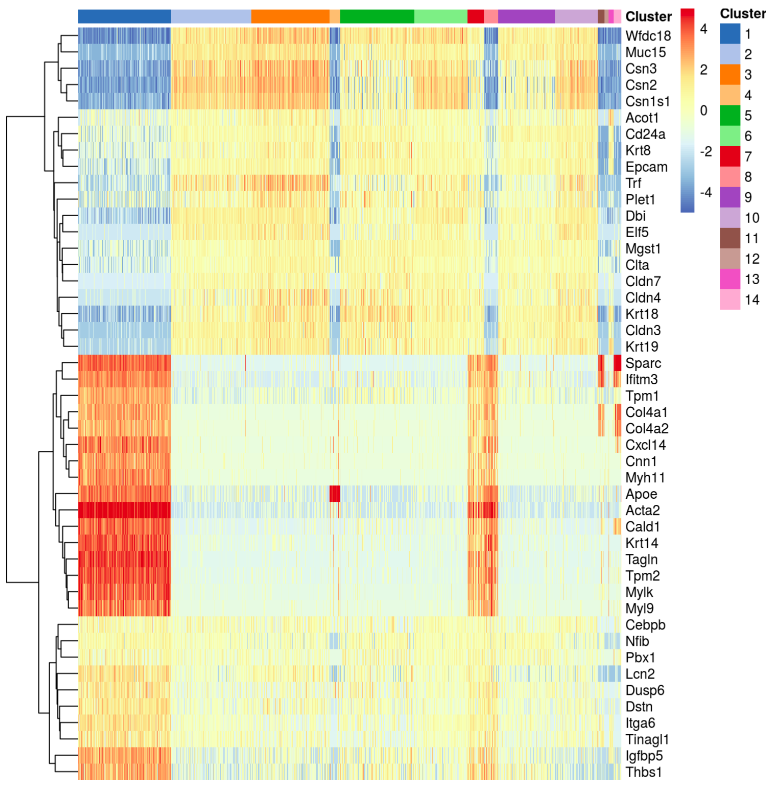


Figure 3: Heatmap of mean-centred and normalized log-expression values for the top set of markers for cluster 7 in the 416B dataset. Column colours represent the cluster to which each cell is assigned, as indicated by the legend.

Closer examination of some known markers suggests that the offending cluster consists of doublets of basal cells (*Acta2*) and alveolar cells (*Csn2*) (Figure 4). Indeed, no cell type is known to strongly express both of these genes at the same time, which supports the hypothesis that this cluster consists solely of doublets. Of course, it is possible that this cluster represents an entirely novel cell type, though the presence of doublets provides a more sober explanation for its expression profile.

```
plotExpression(sce, features=c("Acta2", "Csn2"),
    x="Cluster", colour_by="Cluster")
```

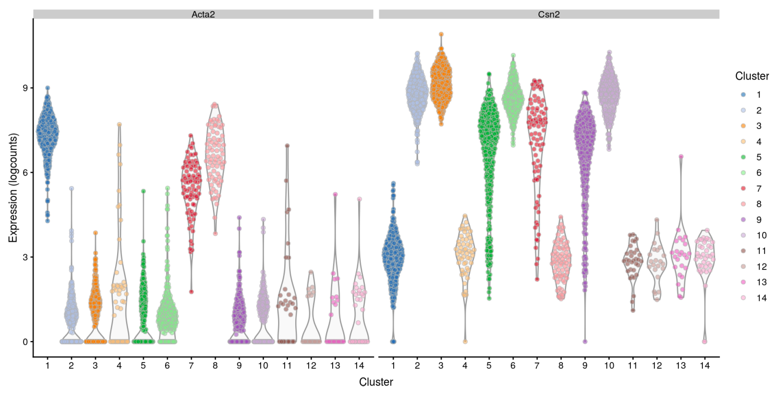


Figure 4: Distribution of log-normalized expression values for *Acta2* and *Csn2* in each cluster. Each point represents a cell.

The strength of `doubletCluster()` lies in its simplicity and ease of interpretation. Suspect clusters can be quickly flagged for further investigation, based on the metrics returned by the function. However, it is obviously dependent on the quality of the clustering. Clusters that are too coarse will fail to separate doublets from other cells, while clusters that are too fine will complicate interpretation.

Comments from Aaron:

- The output of `doubletClusters()` should be treated as a prioritization of "high-risk" clusters that require more careful investigation. We do not recommend using a fixed threshold on any of the metrics to identify doublet clusters. Any appropriate threshold for the metrics will depend on the quality of the clustering and the biological context.
- The pair of parents for each query cluster are identified solely on the basis of the lowest `N`. This means that any `lib.size*` above unity is not definitive evidence against a doublet identity for a query cluster. It is possible for the "true" parent clusters to be adjacent to the detected parents but with slightly higher `N`. If this occurs, inspect the `all.pairs` field for statistics on all possible parent pairs for a given query cluster.
- Clusters with few cells are implicitly more likely to be detected as doublets. This is because they will have less

power to detect DE genes and thus the value of `N` is more likely to be small. Fortunately for us, this is a desirable effect as doublets should be rare in a properly performed scRNA-seq experiment.

4 Doublet detection by simulation

4.1 Background

The other doublet detection strategy involves *in silico* simulation of doublets from the single-cell expression profiles (Dahlin et al. 2018). This is performed using the `doubletCells()` function from *scran (https://bioconductor.org/packages/3.8/scran)*, which will:

1. Simulate thousands of doublets by adding together two randomly chosen single-cell profiles.
2. For each original cell, compute the density of simulated doublets in the surrounding neighbourhood.
3. For each original cell, compute the density of other observed cells in the neighbourhood.
4. Return the ratio between the two densities as a “doublet score” for each cell.

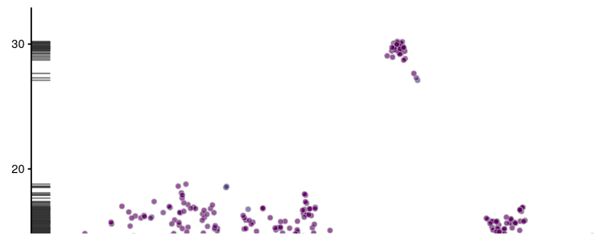
This approach assumes that the simulated doublets are good approximations for real doublets. The use of random selection accounts for the relative abundances of different subpopulations, which affect the likelihood of their involvement in doublets; and the calculation of a ratio avoids high scores for non-doublet cells in highly abundant subpopulations. We see the function in action below:

```
set.seed(100)
dbl.dens <- doubletCells(sce, approximate=TRUE)
summary(dbl.dens)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.00000 0.01771 0.07163 0.15626 0.13794 7.55793
```

The highest doublet scores are concentrated in a single cluster of cells in the centre of Figure 5.

```
sce$DoubletScore <- dbl.dens
plotTSNE(sce, colour_by="DoubletScore")
```



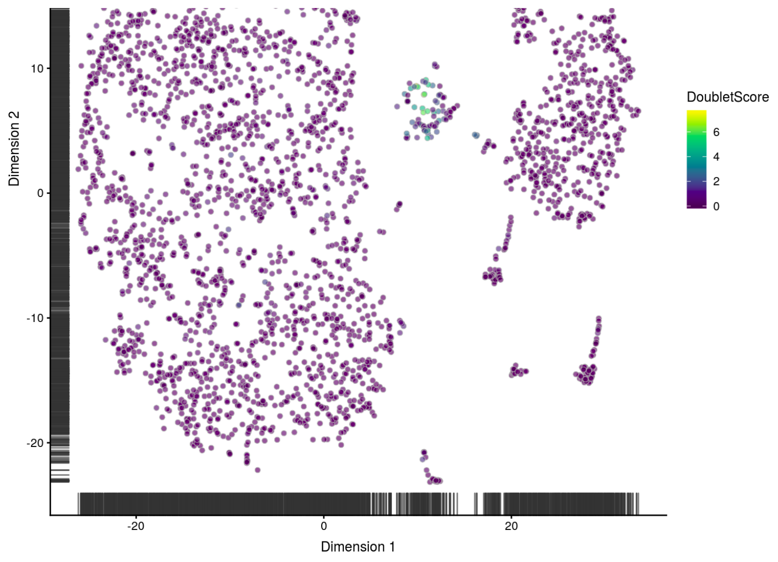


Figure 5: t-SNE plot of the mammary gland data set. Each point is a cell coloured according to its doublet density.

From the clustering information, we see that the affected cells belong to the same cluster that was identified using `doubletCluster()` (Figure 6).

```
plotColData(sce, x="Cluster", y="DoubletScore", colour_by="C
luster")
```

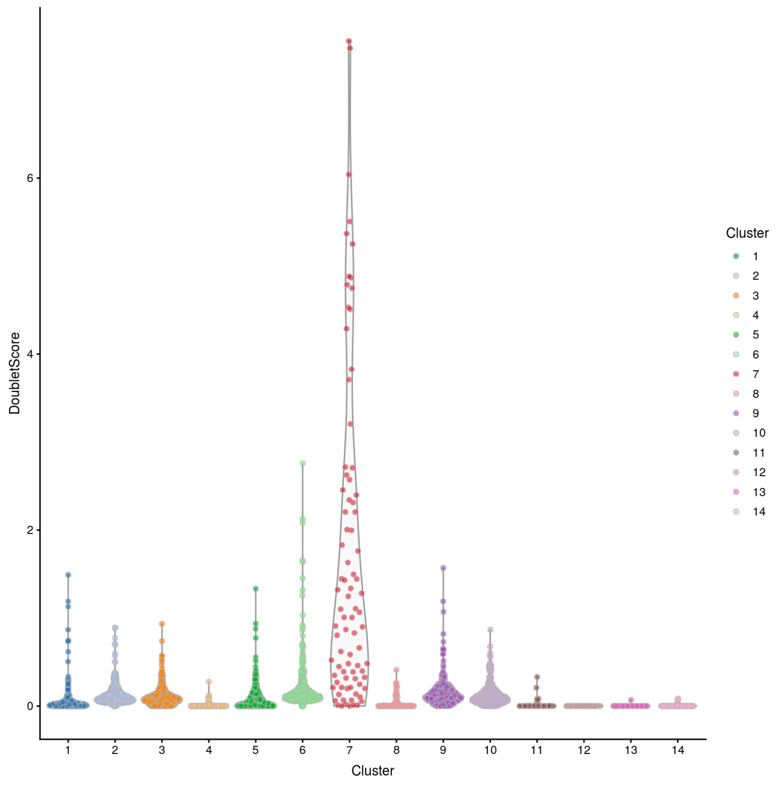


Figure 6: Distribution of doublet scores for each cluster in the mammary gland data set. Each point is a cell.

The advantage of `doubletCells()` is that it does not depend on clusters, reducing the sensitivity of the results to clustering quality. The downside is that it requires some strong assumptions about how doublets form, such as the combining proportions and the sampling from pure subpopulations. In particular, `doubletCells()` treats the library size of each cell as an accurate proxy for its total RNA content. If this is not true, the simulation will not combine expression profiles from different cells in the correct proportions. This means that the simulated doublets will be systematically shifted away from the real doublets, resulting in doublet scores that are too low.

It should also be stressed that the interpretation of the doublet scores is relative. It is difficult to generally define a fixed threshold above which libraries are to be considered doublets. One strategy is to define a threshold at a percentile corresponding to the expected proportion of doublets, based on experimental knowledge of the number of loaded cells. Another approach is to use these scores in the context of cluster annotation, where clusters with higher-than-average doublet scores are considered suspect.

Comments from Aaron:

- To speed up the density calculations, `doubletCells()` will perform a PCA on the log-expression matrix. When `approximate=TRUE`, methods from the *irlba (https://CRAN.R-project.org/package=irlba)* package are used to perform a fast approximate PCA. This involves randomization so it is necessary to call `set.seed()` to ensure that results are reproducible.
- In some cases, we can improve the clarity of the result by setting `force.match=TRUE` in the `doubletCells()` call. This will forcibly match each simulated doublet to the nearest neighbouring cells in the original data set. Any systematic differences between simulated and real doublets will be removed, provided that the former are close enough to the latter to identify the correct nearest neighbours. This overcomes some of issues related to RNA content but is a rather aggressive strategy that may incorrectly inflate the reported doublet scores.
- The issue of unknown combining proportions can be solved completely if spike-in information is available, e.g., in plate-based protocols. This will provide an accurate estimate of the total RNA content of each cell. To this end, size factors from `computeSpikeFactors()` (see here (https://bioconductor.org/packages/3.8/simpleSingleCell/vignettes/xtra-2-spike)) can be supplied to the `doubletCells()` function via the `size.factors.content=` argument. This will use the spike-in size factors to scale the contribution of each cell to a doublet library.

5 Concluding remarks

Doublet detection procedures should only be applied to libraries generated in the same experimental batch. It is obviously impossible for doublets to form between two cells that were captured separately. Thus, some understanding of the experimental design is required prior to the use of the above functions. This avoids unnecessary concerns about the validity of clusters when they cannot possibly consist of doublets.

It is also difficult to interpret doublet predictions in data containing cellular trajectories. By definition, cells in the middle of a trajectory are always intermediate between other cells and are liable to be incorrectly detected as doublets. Some protection is provided by the non-linear nature of many real trajectories, which reduces the risk of simulated doublets coinciding with real cells in `doubletCells()`. One can also put more weight on the relative library sizes in `doubletCluster()` instead of relying on `N`, under the assumption that sudden spikes in RNA content are unlikely in a continuous biological process.

6 Session information

We save the `SingleCellExperiment` object with its associated data to file for future use.

```
saveRDS(sce, file="mammary.rds")
```

All software packages used in this workflow are publicly available from the Comprehensive R Archive Network (https://cran.r-project.org (https://cran.r-project.org)) or the Bioconductor project (http://bioconductor.org (http://bioconductor.org)). The specific version numbers of the packages used are shown below, along with the version of the R installation.

```
sessionInfo()
```

```
## R version 3.5.2 (2018-12-20)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.5 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.8-bioc/R/lib/libRblas.so
## LAPACK: /home/biocbuild/bbs-3.8-bioc/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices uti
ls     datasets
## [8] methods   base
##
## other attached packages:
##  [1] Matrix_1.2-15
##  [2] org.Hs.eg.db_3.7.0
##  [3] EnsDb.Hsapiens.v86_2.99.0
##  [4] ensembldb_2.6.3
##  [5] AnnotationFilter_1.6.0
##  [6] DropletUtils_1.2.2
##  [7] pheatmap_1.0.12
##  [8] cluster_2.0.7-1
##  [9] dynamicTreeCut_1.63-1
## [10] limma_3.38.3
## [11] scran_1.10.2
## [12] scater_1.10.1
## [13] ggplot2_3.1.0
## [14] TxDb.Mmusculus.UCSC.mm10.ensGene_3.4.0
## [15] GenomicFeatures_1.34.1
## [16] org.Mm.eg.db_3.7.0
## [17] AnnotationDbi_1.44.0
## [18] SingleCellExperiment_1.4.1
## [19] SummarizedExperiment_1.12.0
## [20] DelayedArray_0.8.0
## [21] BiocParallel_1.16.5
## [22] matrixStats_0.54.0
## [23] Biobase_2.42.0
## [24] GenomicRanges_1.34.0
## [25] GenomeInfoDb_1.18.1
## [26] IRanges_2.16.0
## [27] S4Vectors_0.20.1
## [28] BiocGenerics_0.28.0
## [29] bindrcpp_0.2.2
## [30] BiocFileCache_1.6.0
## [31] dbplyr_1.2.2
## [32] knitr_1.21
## [33] BiocStyle_2.10.0
##
## loaded via a namespace (and not attached):
##  [1] ProtGenerics_1.14.0      bitops_1.0-6
##  [3] bit64_0.9-7              RColorBrewer_1.1-2
##  [5] progress_1.2.0           httr_1.4.0
##  [7] tools_3.5.2              irlba_2.3.2
##  [9] R6_2.3.0                 KernSmooth_2.23-15
## [11] HDF5Array_1.10.1         vipor_0.4.5
## [13] DBI_1.0.0                lazyeval_0.2.1
## [15] colorspace_1.3-2         withr_2.1.2
## [17] tidyselect_0.2.5         gridExtra_2.3
## [19] prettyunits_1.0.2        bit_1.1-14
## [21] curl_3.2                 compiler_3.5.2
## [23] BiocNeighbors_1.0.0      rtracklayer_1.42.1
## [25] labeling_0.3             bookdown_0.9
## [27] scales_1.0.0             rappdirs_0.3.1
## [29] stringr_1.3.1            digest_0.6.18
## [31] Rsamtools_1.34.0         rmarkdown_1.11
## [33] XVector_0.22.0           pkgconfig_2.0.2
## [35] htmltools_0.3.6          highr_0.7
## [37] rlang_0.3.1              RSQLite_2.1.1
## [39] DelayedMatrixStats_1.4.0 bindr_0.1.1
## [41] dplyr_0.7.8              RCurl_1.95-4.11
## [43] magrittr_1.5             GenomeInfoDbData_1.2.0
## [45] Rcpp_1.0.0               ggbeeswarm_0.6.0
## [47] munsell_0.5.0            Rhdf5lib_1.4.2
## [49] viridis_0.5.1            edgeR_3.24.3
## [51] stringi_1.2.4            yaml_2.2.0
## [53] zlibbioc_1.28.0          Rtsne_0.15
## [55] rhdf5_2.26.2             plyr_1.8.4
## [57] grid_3.5.2               blob_1.1.1
## [59] crayon_1.3.4             lattice_0.20-38
## [61] Biostrings_2.50.2        cowplot_0.9.4
## [63] hms_0.4.2                locfit_1.5-9.1
## [65] pillar_1.3.1             igraph_1.2.2
## [67] reshape2_1.4.3           biomaRt_2.38.0
## [69] XML_3.98-1.16            glue_1.3.0
## [71] evaluate_0.12            BiocManager_1.30.4
## [73] gtable_0.2.0             purrr_0.2.5
## [75] assertthat_0.2.0         xfun_0.4
## [77] viridisLite_0.3.0        tibble_2.0.0
## [79] GenomicAlignments_1.18.1 beeswarm_0.2.3
## [81] memoise_1.1.0            statmod_1.4.30
```

References

Bach, K., S. Pensa, M. Grzelak, J. Hadfield, D. J. Adams, J. C. Marioni, and W. T. Khaled. 2017. “Differentiation dynamics of mammary epithelial cells revealed by single-cell RNA sequencing.” *Nat Commun* 8 (1):2128.

Dahlin, J. S., F. K. Hamey, B. Pijuan-Sala, M. Shepherd, W. W. Y.

Lau, S. Nestorowa, C. Weinreb, et al. 2018. “A single-cell hematopoietic landscape resolves 8 lineage trajectories and defects in Kit mutant mice.” *Blood* 131 (21):e1–e11.

Huber, W., V. J. Carey, R. Gentleman, S. Anders, M. Carlson, B. S. Carvalho, H. C. Bravo, et al. 2015. “Orchestrating high-throughput genomic analysis with Bioconductor.” *Nat. Methods* 12 (2):115–21.

Kang, H. M., M. Subramaniam, S. Targ, M. Nguyen, L. Maliskova, E. McCarthy, E. Wan, et al. 2018. “Multiplexed droplet single-cell RNA-sequencing using natural genetic variation.” *Nat. Biotechnol.* 36 (1):89–94.

Lun, A. T., K. Bach, and J. C. Marioni. 2016. “Pooling across cells to normalize single-cell RNA sequencing data with many zero counts.” *Genome Biol.* 17 (April):75.

McCarthy, D. J., K. R. Campbell, A. T. Lun, and Q. F. Wills. 2017. “Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R.” *Bioinformatics* 33 (8):1179–86.

Stoeckius, M., S. Zheng, B. Houck-Loomis, S. Hao, B. Yeung, P. Smibert, and R. Satija. 2017. “Cell "Hashing" with Barcoded Antibodies Enables Multiplexing and Doublet Detection for Single Cell Genomics.” *bioRxiv*. Cold Spring Harbor Laboratory. https://doi.org/10.1101/237693 (https://doi.org/10.1101/237693).

Van der Maaten, L., and G. Hinton. 2008. “Visualizing Data Using T-SNE.” *J. Mach. Learn. Res.* 9 (2579-2605):85.

Xu, C., and Z. Su. 2015. “Identification of cell types from single-cell transcriptomes using a novel clustering method.” *Bioinformatics* 31 (12):1974–80.

Zheng, G. X., J. M. Terry, P. Belgrader, P. Ryvkin, Z. W. Bent, R. Wilson, S. B. Ziraldo, et al. 2017. “Massively parallel digital transcriptional profiling of single cells.” *Nat Commun* 8 (January):14049.