Drop-seq Analysis

Martin Holub

January 1, 2018

1 Introduction

Developments in RNA sequencing protocols enabled high-throughput and cost-effective analysis of single cell suspensions prepared from fresh tissues [Macosko et al., 2015, Ziegenhain et al., 2017]. In contrast, similar methods are lacking for clinical samples, archived materials and tissues that cannot be easily dissociated [Habib et al., 2017]. Although the protocols published to date [Habib et al., 2016, Lake et al., 2016, Lacar et al., 2016] enable RNA analyzes from such samples, they do not scale to large number of cells.

Variation of Drop-seq [Macosko et al., 2015] method was presented in the paper "Massively parallel single-nucleus RNA-seq with DroNc-seq" [Habib et al., 2017] that addresses this limitation. Specifically, the EZ PREP Buffer (Sigma Cat #NUC-101) was used together with filtering through 35μm cell-strainer to obtain suspension of single nuclei and the design of microfluidics device was adjusted to account for lower amount of RNA in nuclei compared to cells. The nuclei were then processed according to the Drop-seq protocol [Macosko et al., 2015] (see Figure 1). Briefly, nuclei were coencapsulated with barcoded beads and lysed. The released mRNA hybridized to unique barcodes, droplets were broken, and the beads collected in bulk. The hybridized mRNA was reverse-transcribed, yielding cDNA which was PCR amplified and sequenced. Subsequently, bioinformatic analysis confirmed biological relevance of resulting data by identifying cell-types with graph-based clustering algorithm [Shekhar et al., 2016].

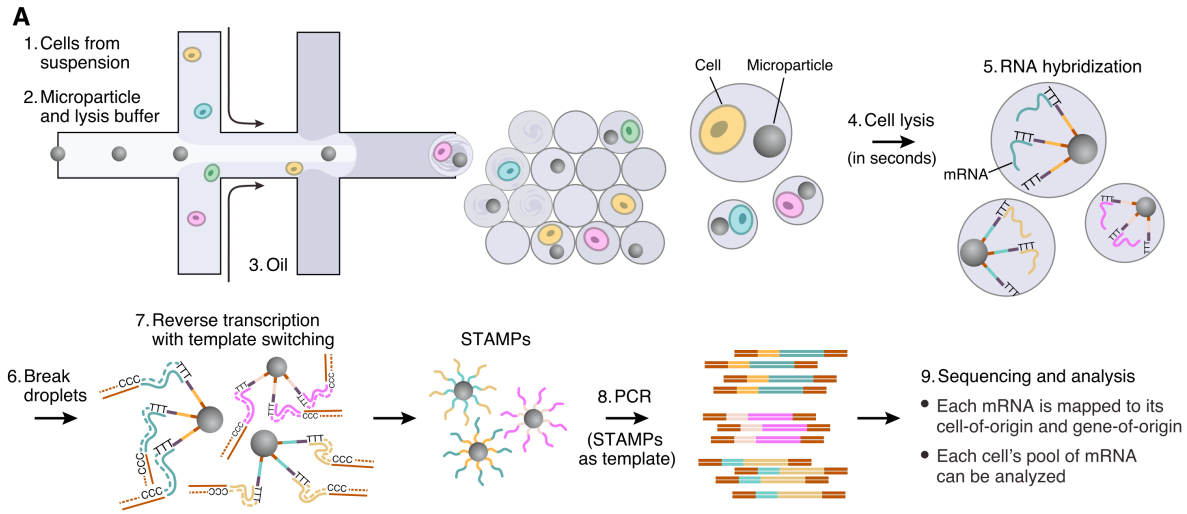


Figure 1: Drop-seq Protocol.

1.1 Scope

In this study, we are interested in reproducing parts of the data analysis from the aforementioned paper. Specifically, we use the raw reads as input and eventually obtain 2-dimensional embedding where we assign class membership to each nucleus based on assignments made available by the authors. We then visually compare resulting plot with plot presented in the paper to asses the level to which we have managed to

reproduce the data analysis. This report aims to be a guide through the analysis, describing individual steps. Should one want to implement similar workflow, this document can serve as a good starting point.

1.2 Overview

There is multitude of steps necessary to obtain the desired dimensionality-reduced plot. Before we delve into details, we consider first the top-level view of the procedure as summarized from the paper:

1. Preprocessing (*command line, cluster*)
    - barcodes filtered for minimum number of transcripts and aggregated (1 edit tolerance)
    - gene counts aggregated on unique UMIs (1 substitution tolerance)
    - reads with low Phred filtered out, trimmed from nucleus barcode (12 bases) and unique molecular index (8 bases)
2. Alignment (*command line, cluster*)
    - to mm10 mouse UCSC reference genome using STAR and recording exonic regions.
3. Export (*command line, cluster*)
    - transcript counts were assembled into digital gene expression (DGE) matrix.
4. Additional Filtering (*R, local*)
    - the DGE was scaled by total number of UMI counts, multiplied per by mean number of transcripts and log transformed
    - effects of number of transcripts and genes detected per nucleus were regressed out
    - nuclei with less than 200 genes were filtered out
    - genes detected in less then 10 nuclei were removed.
5. Finding variable genes, dimensionality reduction and visualization (*R, local*)
    - highly variable genes were selected by fitting gamma-distribution relationship between mean counts and coefficient of variation
    - dimensionality of data was reduced, using the DGE matrix consisting only of highly variable genes
    - most significant principal components in data were chosen based on the largest value of eigenvalue gap
    - 2D nonlinear embedding of the nuclei profiles was generated with tSNE using the previously identified principal components as input.

In the overview, the individual steps were labeled depending on whether they were carried out on cluster or on local machine in our own analysis. This separation was required due to large volumes of data and subsequent high computational complexity of most of the steps. As a consequence, most of the chunks in this report are not evaluated at knitting (`eval = FALSE`), instead precomputed objects are used.

2 Wokrflow

2.1 Working on Cluster

For the computationally complex parts of the analysis, Euler, cluster at ETH, was used.

2.1.1 Connecting

We log to Euler with:

```
ssh USERNAME@euler.ethz.ch
```

2.1.2 Storage

Euler offers its standard users (e.g. students) two places to deposit their data `home` and `scratch`. Briefly, `home` will hold small volumes of data indefinitely while `scratch` holds big data intermittently (see also storage comparsion).

The corresponding paths are as follows:

```
home="/cluster/home/USERNAME"
scratch="/cluster/scratch/USERNAME"
```

and we access them with `$HOME` and `$SCRATCH`.

2.1.3 Modules

Next, we load modules required for our analysis and we do this every time we log to the cluster. To automate this setup, we define `.bashrc` file in the `home` directory:

```
mkdir -p $HOME/python/lib64/python3.6/site-packages
export PYTHONPATH=$HOME/python/lib64/python3.6/site-packages:$PYTHONPATH
export PATH=$PYTHONPATH:$PATH
module load new gcc/4.8.2 python/3.6.0
python -m pip install --install-option="--prefix=$HOME/python" rpy2 multiqc

module load java
module load new gcc/4.8.2 r/3.4.0
module load gcc/4.8.2 star/2.4.2a
module load samtools
```

2.1.4 Job submission

When submitting jobs to cluster nodes, we use variations of the following example commands:

```
# for shell scripts
bsub -J "<jobname>" -w "done(JOBID)" -W 640 -n 8 -R "rusage[mem=8192]" < scriptname.sh
# for R scripts
bsub -W 120 -n 8 -R "rusage[mem=4096]" "R --vanilla --slave < RNA_seq.R > result.out"
```

where (all optional):

- `-W` is the time limit in minutes,
- `-n` is the number of threads
- `-J` is the job name
- `-R` is the memory requirement per thread

2.2 Dependencies

Multiple additional scripts and programs are needed for the preprocessing, alignment and DGE export. Specifically:

- Drop-seq_tools-1.12 & Picard
  - a suite of java scripts for preprocessing of Drop-seq data which we will make extended use of
  - there is a manual available that we will follow closely
  - Drop-seq Tutorial & Troubleshooting then offers additional information on the experimental protocol.

- dropSeqPipe
  - a wrapper for the Drop-seq Tools that provides `yaml` and `python` interface and makes uses of Snakemake to build reproducible workflow.
- FASTQC
  - a general quality-control tool for high-throughput sequencing data.

All these (and potential dependencies) are installed following the instructions at respective websites and placed in `$HOME/Software`.

2.3 Data

In the paper that we follow, authors sequenced archived brain tissue samples, previously obtained from human and mouse species. Due to regulatory restrictions, only the raw reads sequenced from mouse tissues are available. The data was deposited at the Single Cell Portal. We download the data in bulk with:

```
curl "https://portals.broadinstitute.org/single_cell/bulk_data/
      dronc-seq-single-nucleus-rna-seq-on-mouse-archived-brain/all/000000" -o cfg.txt
curl -K cfg.txt -o $SCRATCH
```

Note that the zeros at the end of the link indicate unique key for download that you will have to generate and that this is valid only for 30 minutes.

2.3.1 Rename

For convenience we remove redundant suffix in the file name such that `file_001.fastq.gz` becomes `file.fastq.gz` with:

```
for fname in *.fastq.gz
do
  mv "$fname" "$(echo "$fname" | sed -r 's/_001//')"
done
```

As result, we obtain 20 zipped fastq files in total size of 110 GB.

```
-rw-r----- 1 USER T0000  2276111242 Dec 19 17:23 0_1a_S1_R1.fastq.gz
-rw-r----- 1 USER T0000  6403618432 Dec 19 17:32 0_1a_S1_R2.fastq.gz
-rw-r----- 1 USER T0000  2467148359 Dec 19 17:18 0_1b_S2_R1.fastq.gz
-rw-r----- 1 USER T0000  6945108236 Dec 19 17:19 0_1b_S2_R2.fastq.gz
-rw-r----- 1 USER T0000  8635163524 Dec 19 17:21 0_2a_S1_R1.fastq.gz
-rw-r----- 1 USER T0000 23786888919 Dec 19 17:31 0_2a_S1_R2.fastq.gz
-rw-r----- 1 USER T0000  1348694912 Dec 19 17:21 0_2b_S2_R1.fastq.gz
-rw-r----- 1 USER T0000  3752821903 Dec 19 17:36 0_2b_S2_R2.fastq.gz
-rw-r----- 1 USER T0000  3216586684 Dec 19 17:19 HP1_S1_R1.fastq.gz
-rw-r----- 1 USER T0000  8362169363 Dec 19 17:26 HP1_S1_R2.fastq.gz
-rw-r----- 1 USER T0000  3150012054 Dec 19 17:21 HP3_S1_R1.fastq.gz
-rw-r----- 1 USER T0000  8720716415 Dec 19 17:27 HP3_S1_R2.fastq.gz
-rw-r----- 1 USER T0000  3525694246 Dec 19 17:31 mHP2_S1_R1.fastq.gz
-rw-r----- 1 USER T0000  9775604305 Dec 19 17:36 mHP2_S1_R2.fastq.gz
-rw-r----- 1 USER T0000  1950157698 Dec 19 17:34 mPFC2_S2_R1.fastq.gz
-rw-r----- 1 USER T0000  5428484811 Dec 19 17:33 mPFC2_S2_R2.fastq.gz
-rw-r----- 1 USER T0000  2123092106 Dec 19 17:34 PFC1_S2_R1.fastq.gz
-rw-r----- 1 USER T0000  5496781007 Dec 19 17:24 PFC1_S2_R2.fastq.gz
-rw-r----- 1 USER T0000  2821470681 Dec 19 17:24 PFC3_S2_R1.fastq.gz
-rw-r----- 1 USER T0000  7805172406 Dec 19 17:23 PFC3_S2_R2.fastq.gz
```

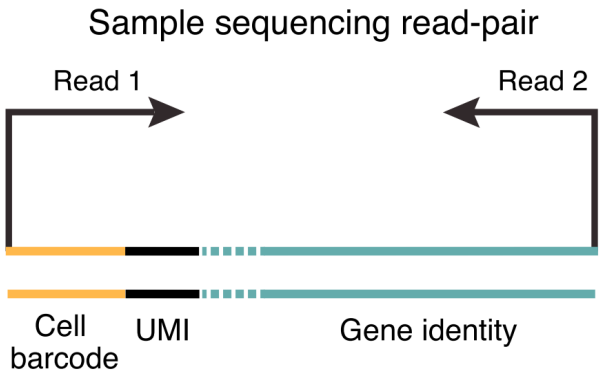


Figure 2: Barcode and gene reads.

This corresponds to 10 samples as each sample requires 2 files `{sample}_R1.fastq.gz` and `{sample}_R2.fastq.gz`. `R1` file holds reads for cell and molecular barcode (also termed UMI, unique molecular identifier) and `R2` file then holds read from actual mRNA sequence (Figure 2 [Macosko et al., 2015]), both files are required for quantification of transcript abundance.

2.4 Reference Index

At later steps of the pipeline, we align the sequenced reads to a reference genome using STAR. For that, we first download the reference genome file:

```
cd $SCRATCH
mkdir reference; cd reference
wget "ftp://ftp.ncbi.nlm.nih.gov/geo/series/GSE63nnn/GSE63472/suppl/
      GSE63472_mm10_reference_metadata.tar.gz"
tar xzvf GSE63472_mm10_reference_metadata.tar.gz
```

Next we call STAR to generate the index:

```
genome_dir="$SCRATCH/reference/mm10/STAR_index"
genome_fasta="$SCRATCH/reference/mm10/mm10.fasta"
genome_gtf="$SCRATCH/reference/mm10/mm10.gtf"
overhang_length=59

mkdir genome_dir
echo "'STAR version:\n'"
STAR --version
STAR --runMode genomeGenerate --runThreadN 8 --genomeDir $genome_dir \
--genomeFastaFiles $genome_fasta --sjdbGTFfile $genome_gtf --sjdbOverhang $overhang_length
```

Where the `overhang_length` was determined based on information given in the paper and corresponds to `read_length` - 1. Note that `overhang_length` must be identical for the construction of genome index and the alignment.

2.5 Preprocessing

Recall that, as our major milestone, we seek the digital expression matrix (DGE) summarizing data from all the raw fastq files (both genomic and barcode reads). Figure 3 shows simplified visual representation of the necessary procedure.

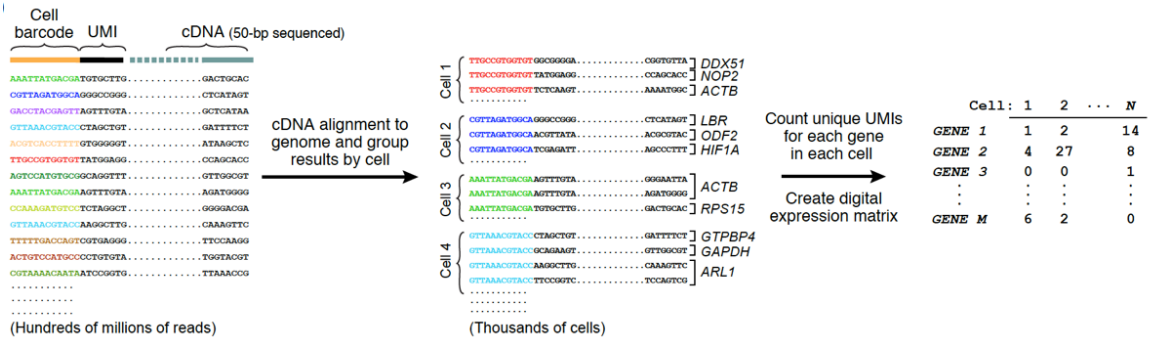


Figure 3: Preprocessing overview [Macosko et al., 2015].

Drop-seq Tools is a set of command line programs for preprocessing and alignment of Drop-seq (and DroNc-seq) data running in java. These scripts are wrapped by dropSeqPipe into reproducible pipeline implemented in Python using Snakemake. We control the pipeline behavior by two configuration files, `config.yaml` and `local.yaml`, content of which is decried in following.

The `config.yaml` file is:

```
Samples:
    0_2a_S1:
        fraction: 0.07
        expected_cells: 1400
    0_1b_S2:
        fraction: 0.07
        expected_cells: 1400
    0_2b_S2:
        fraction: 0.07
        expected_cells: 1400
    0_1a_S1:
        fraction: 0.07
        expected_cells: 1400
    mPFC2_S2:
        fraction: 0.07
        expected_cells: 1400
    PFC1_S2:
        fraction: 0.07
        expected_cells: 1400
    PFC3_S2:
        fraction: 0.07
        expected_cells: 1400
    HP3_S1:
        fraction: 0.07
        expected_cells: 1400
    HP1_S1:
        fraction: 0.07
        expected_cells: 1400
    mHP2_S1:
        fraction: 0.07
        expected_cells: 1400
GENOMEREF: $SCRATCH/reference/mm10/mm10.fasta
```

```
REFFLAT: $SCRATCH/reference/mm10/mm10.refFlat
METAREF: $SCRATCH/reference/mm10/STAR_index
RRNAINTERVALS: $SCRATCH/reference/mm10/mm10.rRNA.intervals
GTF: $SCRATCH/reference/mm10/mm10.gtf
CORES: 12
SPECIES:
    - MOUSE
GLOBAL:
    5PrimeSmartAdapter: AAGCAGTGGTATCAACGCAGAGT
    data_type: singleCell
    allowed_aligner_mismatch: 10
    min_count_per_umi: 1
    min_umis_per_cell: 2
    min_genes_per_cell: 200
    min_reads_per_cell: 10000
    Cell_barcode:
        start: 1
        end: 12
        min_quality: 10
        num_below_quality: 1
    UMI:
        start: 13
        end: 20
        min_quality: 10
        num_below_quality: 1
```

Where parameters in `Samples` section were determined based on experimental protocol. Specifically, the `fraction` is given by the number of nuclei per ml (300'000) and number of droplets per ml (4'500'000) giving Poisson loading parameter of 0.07.

The expected number of cells can be determined from `knee plot` (Figure 4, note that this one is available only aposteriori, after alignment and initial preprocessing steps). Ideally, we can identify an inflection point beyond which STAMPs (single-cell transcriptomes attached to microparticles) represent droplets only with barcoded beads, and no cells coencapsulated in them. Alternatively, we can make use of the fact that the experimental protocol mentions that libraries were sample from a pool of 20'000 STAMPs, which, given the Poisson loading parameter, yields 1'400 as the expected number of cells.

Parameters in `GLOBAL` were selected based on information from Drop-seq manual and the Methods section of the paper in study and will be described bellow for workflow steps that use them.

The `local.yaml` file is:

```
TMPDIR: $SCRATCH/temp
PICARD: $HOME/Software/Drop-seq_tools-1.12/3rdParty/picard/picard.jar
DROPSEQ: $HOME/Software/Drop-seq_tools-1.12
STAREXEC: /cluster/apps/star/2.4.2a/x86_64/STAR
FASTQCEXEC: $HOME/Software/FastQC/fastqc
CORES: 8
READLENGTH: 59
```

Where read length must be consistent with STAR index for all samples.

With these files defined, we invoke the dropSeqPipe with

```
$HOME/python/bin/dropSeqPipe -f $SCRATCH -c $HOME/local.yaml -m <mode>
```

where `<mode>` is subsequently:

- `fastqc` to perform quality control,

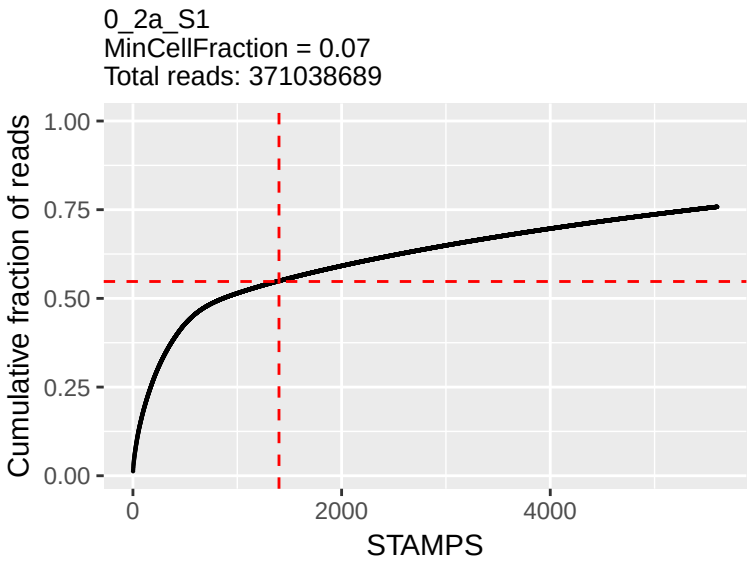


Figure 4: Knee plot.

- `pre-process` to go from raw reads to aligned and sorted data,
- `generate-plots` to genreate knee plots and do MultiQC,
- `extract-expression` to summarize data to digital expression matrix (DGE).

In the following, we go over individual dropSeqPipe/Drop-seq Tools steps. The sections are named by the corresponding Drop-seq Tool or the name of the Picard's function. Parameters in curly braces or those given in capitals correspond to definitions in the two `.yaml` configuration files listed above.

2.5.1 BCL to FASTQ

In this step, `BCL` files were converted to `FASTQ` files. This can be done for example with bcl2fastq. As we have directly downloaded the `.fastq` files, we omit it.

2.5.2 FastqToSam

Converts raw input files from `{sample}.fastq.gz` to `{sample}_unaligned.bam` using Picard's `FastqToSam`.

```
rule fastq_to_sam:
    """Create an empty bam file linking cell/UMI barcodes to reads"""
    input:
        r1='{sample}_R1.fastq.gz',
        r2='{sample}_R2.fastq.gz'
    output:
        temp('{sample}_unaligned.bam')
    threads: CORES
    shell:
        """java -Djava.io.tmpdir={TMPDIR} -Xmx8g -Xms4096m -XX:ParallelGCThreads={CORES}\
        -jar {PICARD} FastqToSam\
        F1={input.r1}\
        F2={input.r2}\
        SM=DS O={output}"""
```

2.5.3 TagBamWithReadSequenceExtended

As described, the DroNc-seq protocol sequences paired-end reads of different length. As a result we end up with 2 files for each sample, one holds barcodes, the other the actual RNA sequence. In our case the shorter 20-bp sequence corresponds to the cell barcode (base 1-12) and molecular barcode (base 13-20) whereas the longer 60-bp sequence is the mRNA corresponding to genomic regions.

In the previous step, these two files were aggregated into one. In this step, we then tag the molecular and cell barcodes producing `{sample}_tagged_unmapped.bam` file. The script is called twice, once for cell and once for molecular barcode. If more than `num_below_quality` bases have Phred score bellow `min_quality`, the read pair is flagged.

```
rule stage1:
    input: '{sample}_unaligned.bam'
    output: '{sample}_tagged_unmapped.bam'
    params:
        BC_summary = 'logs/{sample}_CELL_barcode.txt',
        UMI_summary = 'logs/{sample}_UMI_barcode.txt',
        start_trim = 'logs/{sample}_start_trim.txt',
        polyA_trim = 'logs/{sample}_polyA_trim.txt',
        BC_start = config['GLOBAL']['Cell_barcode']['start'],
        BC_end = config['GLOBAL']['Cell_barcode']['end'],
        BC_min_quality = config['GLOBAL']['Cell_barcode']['min_quality'],
        BC_min_quality_num = config['GLOBAL']['Cell_barcode']['num_below_quality'],
        UMI_start = config['GLOBAL']['UMI']['start'],
        UMI_end = config['GLOBAL']['UMI']['end'],
        UMI_min_quality = config['GLOBAL']['UMI']['min_quality'],
        UMI_min_quality_num = config['GLOBAL']['Cell_barcode']['num_below_quality'],
        SmartAdapter = config['GLOBAL']['5PrimeSmartAdapter']
    threads: CORES
    shell:
        """{DROPSEQ}/TagBamWithReadSequenceExtended\
        SUMMARY={params.BC_summary}\
        BASE_RANGE={params.BC_start}-{params.BC_end}\
        BASE_QUALITY={params.BC_min_quality}\
        BARCODED_READ=1\
        DISCARD_READ=false\
        TAG_NAME=XC\
        NUM_BASES_BELOW_QUALITY={params.BC_min_quality_num}\
        INPUT={input}\
        OUTPUT=/dev/stdout COMPRESSION_LEVEL=0 |\
        \
        {DROPSEQ}/TagBamWithReadSequenceExtended\
        SUMMARY={params.UMI_summary}\
        BASE_RANGE={params.UMI_start}-{params.UMI_end}\
        BASE_QUALITY={params.UMI_min_quality}\
        BARCODED_READ=1\
        DISCARD_READ=true\
        TAG_NAME=XM\
        NUM_BASES_BELOW_QUALITY={params.UMI_min_quality_num}\
        INPUT=/dev/stdin\
        OUTPUT=/dev/stdout COMPRESSION_LEVEL=0 |\
        \
        ...
        \
```

```
...
\
....
"""
```

Where ... indicate calls explained in the three sections bellow.

2.5.4 FilterBAM

In this step, the read pairs that were flagged as low quality are discarded to produce `{sample}_tagged_unmapped_filtered.bam` file.

...

```
{DROPSEQ}/FilterBAM TAG_REJECT=XQ\
        INPUT=/dev/stdin\
        OUTPUT=/dev/stdout COMPRESSION_LEVEL=0 |\
```

...

2.5.5 TrimStartingSequence

Here we trim away possible traces of Ilumina Nextera SMART adapter from the 5' end. If there is at least 5 contiguous bases with no mismatch to the indicated `5PrimeSmartAdapter` sequence, they are clipped off the read. We apply the knowledge that the barcode has 20 bp where the first 12 bp correspond to cell barcode and the remaining 8 bp to molecular barcode (UMI).

...

```
{DROPSEQ}/TrimStartingSequence\
        OUTPUT_SUMMARY={params.start_trim}\
        SEQUENCE={params.SmartAdapter}\
        MISMATCHES=0\
        NUM_BASES={starttrim_length}\
        INPUT=/dev/stdin\
        OUTPUT=/dev/stdout COMPRESSION_LEVEL=0 |\
```

...

2.5.6 PolyATrimmer

Next, we trim away sequences from the 3' end that consist of at least 6 contiguous A's with no mismatches. This corresponds to the polyadenylated 3' end of mRNA.

...

```
{DROPSEQ}/PolyATrimmer\
        OUTPUT_SUMMARY={params.polyA_trim}\
        MISMATCHES=0\
        NUM_BASES=6\
        OUTPUT={output}\
        INPUT=/dev/stdin
```

2.5.7 SamToFastq

We then run Picard's function `SamToFastq` to convert the filtered files back to `fastq.gz` file format.

```
rule sam_to_fastq:
    input: '{sample}_tagged_unmapped.bam'
    output: '{sample}_tagged_unmapped.fastq.gz'
    threads: CORES
    shell:
        """java -Xmx8g -Xms4096m -XX:ParallelGCThreads={CORES} \
        -jar -Djava.io.tmpdir={TMPDIR}  {PICARD} SamToFastq\
        INPUT={input}\
        FASTQ=/dev/stdout COMPRESSION_LEVEL=0|\
        gzip > {output}"""
```

Alignement

At this point we are ready to align our reads to reference genome. There are multiple tools do this and also alignment-free quantification may be an attractive possibility. In our case, however, we follow the procedure described in the paper and use STAR.

We obtain the alignement with the following `snake` / `bash` snippet:

```
# Configfile
configfile: 'config.yaml'

STAREXEC = config['STAREXEC']
METAREF = config['METAREF']
CORES = config['CORES']
GTF = config['GTF']
MISMATCH = config['GLOBAL']['allowed_aligner_mismatch']
READLENGTH = config['READ_LENGTH']

rule STAR_align:
    input:  '{sample}_tagged_unmapped.fastq.gz'
    output: sam = 'logs/{sample}.Aligned.out.sam'
    params:
        prefix = '{sample}.',
        mismatch = MISMATCH,
        mean_read_length = READLENGTH
    threads: CORES
    shell:"""{STAREXEC}\
            --genomeDir {METAREF}\
            --sjdbGTFfile {GTF}\
            --readFilesCommand zcat\
            --runThreadN {CORES}\
            --readFilesIn {input}\
            --outFileNamePrefix logs/{params.prefix}\
            --sjdbOverhang {params.mean_read_length}\
            --twopassMode Basic\
            --outFilterScoreMinOverLread 0.3\
            --outFilterMatchNminOverLread 0\
            --outFilterMismatchNoverLmax 0.3"""
```

We then continue in the preprocessing pipeline.

2.5.8 SortSam

A Picard tool to sort the `{sample}.Aligned.out.sam` file in queryname order and convert to binary format.

```
rule sort:
    input:
        samples = '{sample}.Aligned.sam'
    output: temp('{sample}_Aligned_sorted.sam')
    threads: CORES
    shell:
        """java -Djava.io.tmpdir={TMPDIR} -Dsamjdk.buffer_size=131072 \
        -XX:GCTimeLimit=50 -XX:GCHeapFreeLimit=10 \
        -XX:ParallelGCThreads={CORES} -Xmx8g -Xms4096m -jar {PICARD} SortSam\
        INPUT={input}\
        OUTPUT={output}\
        SORT_ORDER=queryname\
        TMP_DIR={TMPDIR}"""
```

2.5.9 MergeBamAlignment

A Picard tool to merge the sorted STAR alignment with unaligned BAM file that has been previously tagged with molecular and cell barcodes. This recovers the read identity. Only primary alignments are considered.

```
rule stage3:
    input:  unmapped = '{sample}_tagged_unmapped.bam',
            mapped = '{sample}_Aligned_sorted.sam'
    output: temp('{sample}_gene_exon_tagged.bam')
    threads: CORES
    shell:
        """java -Djava.io.tmpdir={TMPDIR} -Xmx8g -Xms4096m -XX:ParallelGCThreads={CORES}\
        -jar {PICARD} MergeBamAlignment\
        REFERENCE_SEQUENCE={GENOMEREF}\
        UNMAPPED_BAM={input.unmapped}\
        ALIGNED_BAM={input.mapped}\
        INCLUDE_SECONDARY_ALIGNMENTS=false\
        PAIRED_RUN=false\
        OUTPUT=/dev/stdout COMPRESSION_LEVEL=0|\

        ...

        """
```

Where `...` indicates script explained in following section.

2.5.10 TagReadWithGeneExon

For the analysis of DroNc-seq data we wish to retain only reads that map to exonic regions of the genome. Following script tags such reads with "GE" BAM tag using annotation defined in the `mm10.gtf` (or equivalently `mm10.refFlat`) file.

...

```
export _JAVA_OPTIONS="-XX:ParallelGCThreads=8"
  {DROPSEQ}/TagReadWithGeneExon\
  OUTPUT={output}\
  INPUT=/dev/stdin\
  ANNOTATIONS_FILE={REFFLAT}\
  TAG=GE\
  CREATE_INDEX=true
```

2.5.11 DetectBarcodeSynthesisErrors

The program checks the identifier sequence (cell + molecular barcode). Specifically, it looks at the distribution skew in UMIs (molecular barcode) associated with given cell barcode that could have been caused by incomplete synthesis of the cell barcode (e.g. length of only 11 bases), that would in turn result in high percentage of T's at the last UMI position (see Figure 5).

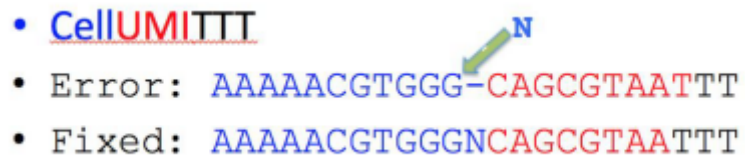


Figure 5: Cell and Molecular Barcode Illustration.

Also, possible matches with any of the PCR primers are checked and, if found, the barcodes and corresponding reads are dropped.

```
rule bead_errors_metrics:
    input: '{sample}_gene_exon_tagged.bam'
    output: '{sample}_final.bam'
    threads: CORES
    params:
        out_stats = 'logs/{sample}_synthesis_stats.txt',
        summary = 'logs/{sample}_synthesis_stats_summary.txt',
        barcodes = lambda wildcards: config['Samples'][wildcards.sample]['expected_cells'] * 2,
        cells = lambda wildcards: config['Samples'][wildcards.sample]['expected_cells'],
        metrics = 'logs/{sample}_rna_metrics.txt',
        umis_per_cell = config['GLOBAL']['min_umis_per_cell']
    shell:
        """{DROPSEQ}/DetectBeadSynthesisErrors\
        INPUT={input}\
        OUTPUT={output}\
        OUTPUT_STATS={params.out_stats}\
        SUMMARY={params.summary}\
        NUM_BARCODES={params.barcodes}\
        MIN_UMIS_PER_CELL={params.umis_per_cell}\
        PRIMER_SEQUENCE=AAGCAGTGGTATCAACGCAGAGTAC;

        {DROPSEQ}/SingleCellRnaSeqMetricsCollector\
        INPUT={input}\
        OUTPUT={params.metrics}\
        ANNOTATIONS_FILE={REFFLAT}\
        NUM_CORE_BARCODES={params.cells}\
        RIBOSOMAL_INTERVALS={RRNAINTERVALS}
```

```
    """
```

At this point are the raw data preprocessing and the alignement finished. The reads have been transformed from paired-end to single-end with corresponding cell and molecular barcodes. They have been also filtered and barcodes were error-corrected.

2.6 Extraction of Digital Gene Expression matrix

Now we are ready to extract DGE matrix from the data. This is done by invoking `DigitalExpression` function. To eliminate multi-mapping reads, we require the quality to be at least 10. We collapse UMI barcodes within a Hamming distance of 1 to account for possible errors during amplification. We also extract only reads that map to exonic regions of the genome (all default settings).

In contrast to the procedure described in the paper, we decide to filter on minimal number of genes and reads per cell already in this step. Authors reported doing so only in R, but this depart should not prevent us from reproducing their results. Moreover, the approach we adopt prevents dragging lot of unused data along.

Further, note that in the previous step we have also already introduced a threshold on minimal number of UMIs per cell, again, in accordance to the descriptions in the paper.

The corresponding `snake` snippet looks like this:

```
"""Extract expression fof single species."""

configfile: 'config.yaml'
DROPSEQ = config['DROPSEQ']

rule all:
    input:
        expand('summary/{sample}_expression_matrix.txt', sample=config['Samples']),
        expand('logs/{sample}_umi_per_gene.tsv', sample=config['Samples']),

rule extract_expression:
    input: '{sample}_final.bam'
    output: 'summary/{sample}_expression_matrix.txt.gz'
    params:
        sample = '{sample}',
        cells = lambda wildcards: config['Samples'][wildcards.sample]['expected_cells'],
        count_per_umi = config['GLOBAL']['min_count_per_umi'],
        genes_per_cell = config['GLOBAL']['min_genes_per_cell'],
        reads_per_cell = config['GLOBAL']['min_reads_per_cell']
    shell:
        """{DROPSEQ}/DigitalExpression\
        I={input}\
        O={output}\
        SUMMARY=summary/{params.sample}_dge.summary.txt \
        CELL_BC_FILE=summary/{params.sample}_barcodes.csv \
        MIN_BC_READ_THRESHOLD={params.count_per_umi} \
        MIN_NUM_GENES_PER_CELL={params.genes_per_cell} \
        MIN_NUM_TRANSCRIPTS_PER_CELL={params.umis_per_cell} \
        NUM_CORE_BARCODES={params.cells}"""
```

Resulting is a DGE matrix that we will work with further. We may also extract other useful data, a matrix associating each gene with all corresponding cell and molecular barcodes:

```
rule extract_umi_per_gene:
    input: '{sample}_final.bam'
```

```
    output: 'logs/{sample}_umi_per_gene.tsv'
    params:
        sample = '{sample}'
    shell:
        """{DROPSEQ}/GatherMolecularBarcodeDistributionByGene\
        I={input}\
        O={output}\
        CELL_BC_FILE=summary/{params.sample}_barcodes.csv"""
```

2.7 Additional processing

For the following steps it is possible to move from cluster to local computer.

From the information available in the paper, we deduce that authors used in-house scripts for the most part of the analysis of the data, including the parts done in R. Availability of these is limited to a previous publication [Shekhar et al., 2016], as that one is accompanied by a github repository with example of analysis. It is unclear however, to which extent was the code used for the analysis in this paper.

Other part of the analysis was done using Seurat, an R package for exploration and analysis of single cell RNA-seq data. Importantly, this package already incorporates most of the functionality implemented in the code from the referred publication. It is unclear however, to which extent are corresponding functions equivalent in terms of their results.

To honor reproducibility and also because some parts of the analysis that we aim to reproduce are available only in Seurat, we opt for this package as our workhorse for following steps.

2.7.1 Load, Scale and Normalize & Remove genes with low expression

Setup basepath:

```
basepath <- getwd()
```

Get paths to files and names of all samples:

```
mouse.datafiles <- list.files(path = file.path(basepath, 'summary/27122017_filter'),
                              pattern = "*_expression_matrix.txt$", full.names = TRUE)
mouse.annofiles <- list.files(path = file.path(basepath, 'summary/27122017_filter'),
                              pattern = "*_dge.summary.txt$", full.names = TRUE)
mouse.umigenefiles <- list.files(path = file.path(basepath, 'logs'),
                                 pattern = "*_umi_per_gene.tsv$", full.names = TRUE)
mouse <- list()
mouse$samples <- gsub("(.*)_S[12]_dge.summary.txt$", "\\1", basename(mouse.annofiles))
```

Construct Seurat Object while retaining sample identity

To quote the paper: "The DGE matrix was scaled by total UMI counts, multiplied by the mean number of transcripts (calculated for each data set separately), and the values were log transformed".

From observation and from available documentation, we infer that the scaling by total number of UMI counts happens implicitly and we thus need to specify only the latter two steps. Additionally we filter out genes with low expression.

Quoting: "A gene is considered detected in a cell if it has at least two unique UMIs (transcripts) associated with it. For each analysis, genes were removed that were detected in less than 10 nuclei."

```
mean_counts <- vector(mode = "list", length = length(mouse$samples))
for (i in 1:length(mouse$samples)){
  # Read data from files
```

```
counts <- read.table(mouse.datafiles[i] , sep = "\t", header = TRUE)
anno <- read.table(mouse.annofiles[i] , sep = "\t", header = TRUE)
umi_per_gene <-  read.table(mouse.umigenefiles[i] , sep = "\t", header = TRUE)

# Agregate number of observations, unique umis, and cell barcodes on gene names
#num_obs <- aggregate(Num_Obs ~ Gene, data = umi_per_gene, sum)
num_umis <- aggregate(Molecular_Barcode ~ Gene, data = umi_per_gene, length)
num_cells <- aggregate(Cell.Barcode ~ Gene, data = umi_per_gene, length)

# Prepare TF filter
idx <- setNames((num_umis$Molecular_Barcode > 2) & (num_cells$Cell.Barcode > 10),
                levels(num_cells$Gene))
num_umis <- num_umis$Molecular_Barcode[idx]
idx_match <- idx[match(counts[ , 1], names(idx))]
idx_match[is.na(idx_match)] <- FALSE

# Move cell barcodes and gene names out of matrix
# Assure unqiue barcodes across samples by prepending sample name
rownames(counts) <- counts[ , 1]
counts <- counts[, -1]
colnames(counts) <- paste0(mouse$samples[i], ".", colnames(counts))
counts <- counts[idx_match, ]

rownames(anno) <- paste0(mouse$samples[i], ".", anno[ , 1])
anno <- anno[ ,-1]
anno <- anno[match(colnames(counts), rownames(anno)), ]

# Get mean count
mean_counts[i] <- mean(anno$NUM_TRANSCRIPTS, na.rm = TRUE)
# Get sparse representation of the data
sparse_counts <- Matrix::Matrix(data.matrix(counts), sparse = TRUE)

if (i == 1) {
  # initialize Seurat
  sObj <- Seurat::CreateSeuratObject(raw.data = sparse_counts, project = 'DroNcSeq',
                                     names.delim = ".",
                                     normalization.method = NULL,
                                     min.cells = 10, min.genes = 200)
  sObj <- Seurat::AddMetaData(sObj, anno, colnames(anno))
  sObj@meta.data$orig.ident <- factor(mouse$samples[i])
  sObj <- Seurat::NormalizeData(object = sObj, normalization.method = "LogNormalize",
                                scale.factor = mean_counts[[i]])

} else {
  # merge Seurat
  sObj_temp <- Seurat::CreateSeuratObject(raw.data = sparse_counts,
                                          project = 'DroNcSeq',
                                          names.delim = ".",
                                          normalization.method = NULL,
                                          min.cells = 10, min.genes = 200)
  sObj <- Seurat::AddMetaData(sObj, anno, colnames(anno))
  sObj_temp@meta.data$orig.ident <- factor(mouse$samples[i])
  sObj_temp <- Seurat::NormalizeData(object = sObj_temp,
                                     normalization.method = "LogNormalize",
```

```
                                    scale.factor = mean_counts[[i]])

    sObj <- Seurat::MergeSeurat(sObj, sObj_temp,
                                min.cells = 10, min.genes = 200,
                                do.normalize = TRUE)
    remove(sObj_temp)
  }

  sprintf("Run %d with mean UMI count %.3f finished.", i, mean_counts[[i]])
}
mouse$sObj <- sObj
remove(sObj, counts, sparse_counts, anno, i, num_cells,
       umi_per_gene, idx, idx_match)
save('mouse', file = "mouseObj.RData")
```

This is time consuming for our big dataset. Alternatively, we may load a precomputed object:

```
load_path <- file.path(basepath, 'data/mouseObj2.RData')
load(load_path)
```

Do quick sanity check:

```
library(Matrix)
idxer <- sample(nrow(mouse$sObj@data), 7)
mouse$sObj@data[idxer, 1:7]
```

```
## 7 x 7 sparse Matrix of class "dgCMatrix"
##               0_1a.TGAACTATCATT 0_1a.TACTGGTCCCCC 0_1a.AAGTCTTACACT
## Fcf1                 0.07990377         0.0536283         .
## Tab1                 .                  0.0536283         .
## Gm16054              .                  .                 .
## Grid2ip              .                  .                 .
## 2810468N07Rik        .                  .                 .
## Gm5607               .                  .                 .
## Gm17566              .                  .                 0.0474232
##               0_1a.CGTAGTTATTAG 0_1a.TGAATTTTTTCT 0_1a.TTTCTAATCGGA
## Fcf1                 0.08705494                 .                 .
## Tab1                 .                          .                 .
## Gm16054              .                          .                 .
## Grid2ip              .                          .                 .
## 2810468N07Rik        .                          .                 .
## Gm5607               .                          .                 .
## Gm17566              .                          .                 .
##               0_1a.CAACCTTATGCA
## Fcf1                  .
## Tab1                  0.1015581
## Gm16054               .
## Grid2ip               .
## 2810468N07Rik         .
## Gm5607                .
## Gm17566               .
```

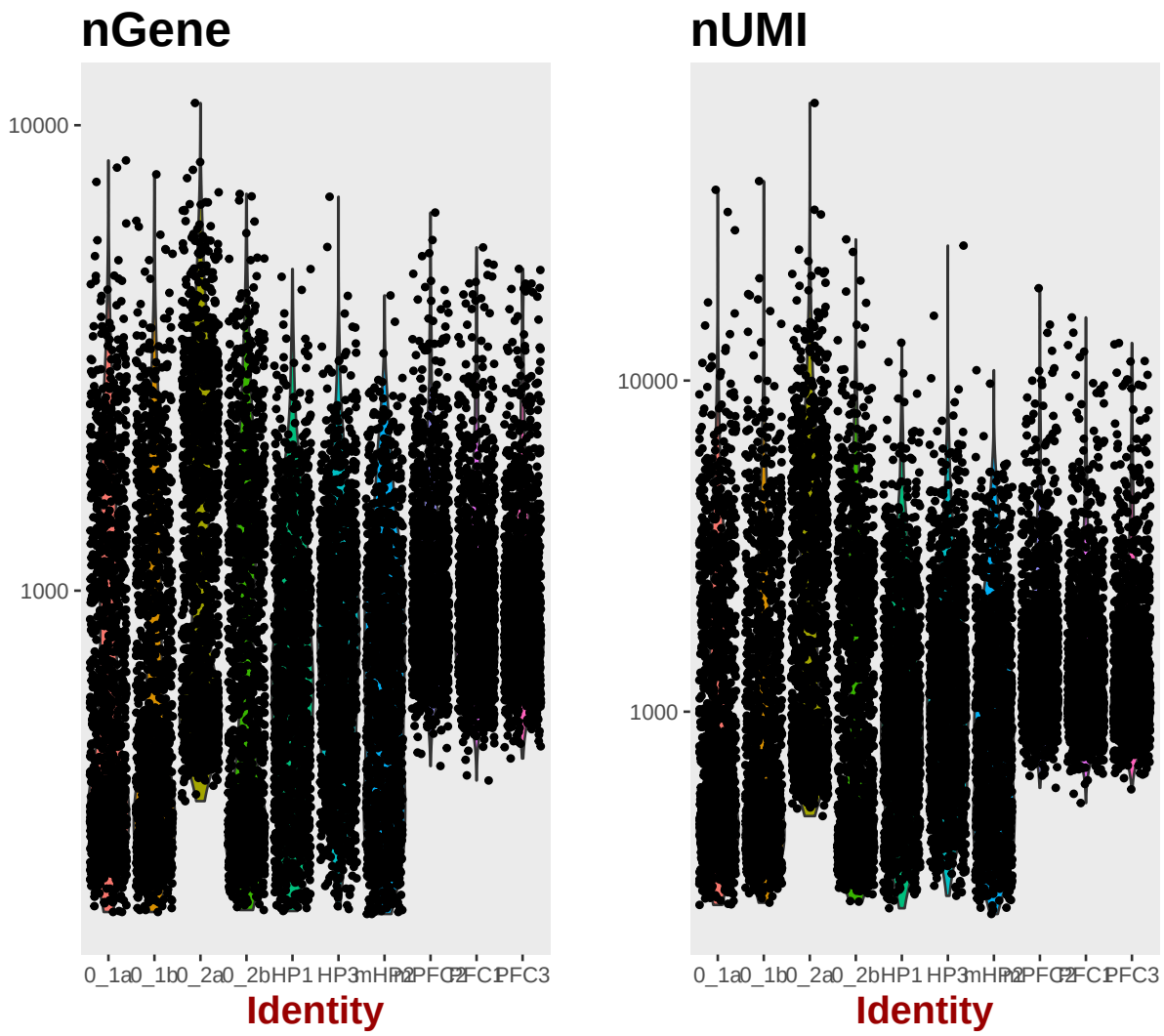
```
mouse$sObj@raw.data[idxer, 1:7]
```

```
## 7 x 7 sparse Matrix of class "dgCMatrix"
##               0_1a.TGAACTATCATT 0_1a.TACTGGTCCCCC 0_1a.AAGTCTTACACT
## Fcf1                         2                  1                 .
```

```
## Tab1                          .                 1                 .
## Gm16054                       .                 .                 .
## Grid2ip                       .                 .                 .
## 2810468N07Rik                 .                 .                 .
## Gm5607                        .                 .                 .
## Gm17566                       .                 .                 1
##               0_1a.CGTAGTTATTAG 0_1a.TGAATTTTTTCT 0_1a.TTTCTAATCGGA
## Fcf1                          1                 .                 .
## Tab1                          .                 .                 .
## Gm16054                       .                 .                 .
## Grid2ip                       .                 .                 .
## 2810468N07Rik                 .                 .                 .
## Gm5607                        .                 .                 .
## Gm17566                       .                 .                 .
##               0_1a.CAACCTTATGCA
## Fcf1                          .
## Tab1                          1
## Gm16054                       .
## Grid2ip                       .
## 2810468N07Rik                 .
## Gm5607                        .
## Gm17566                       .
```

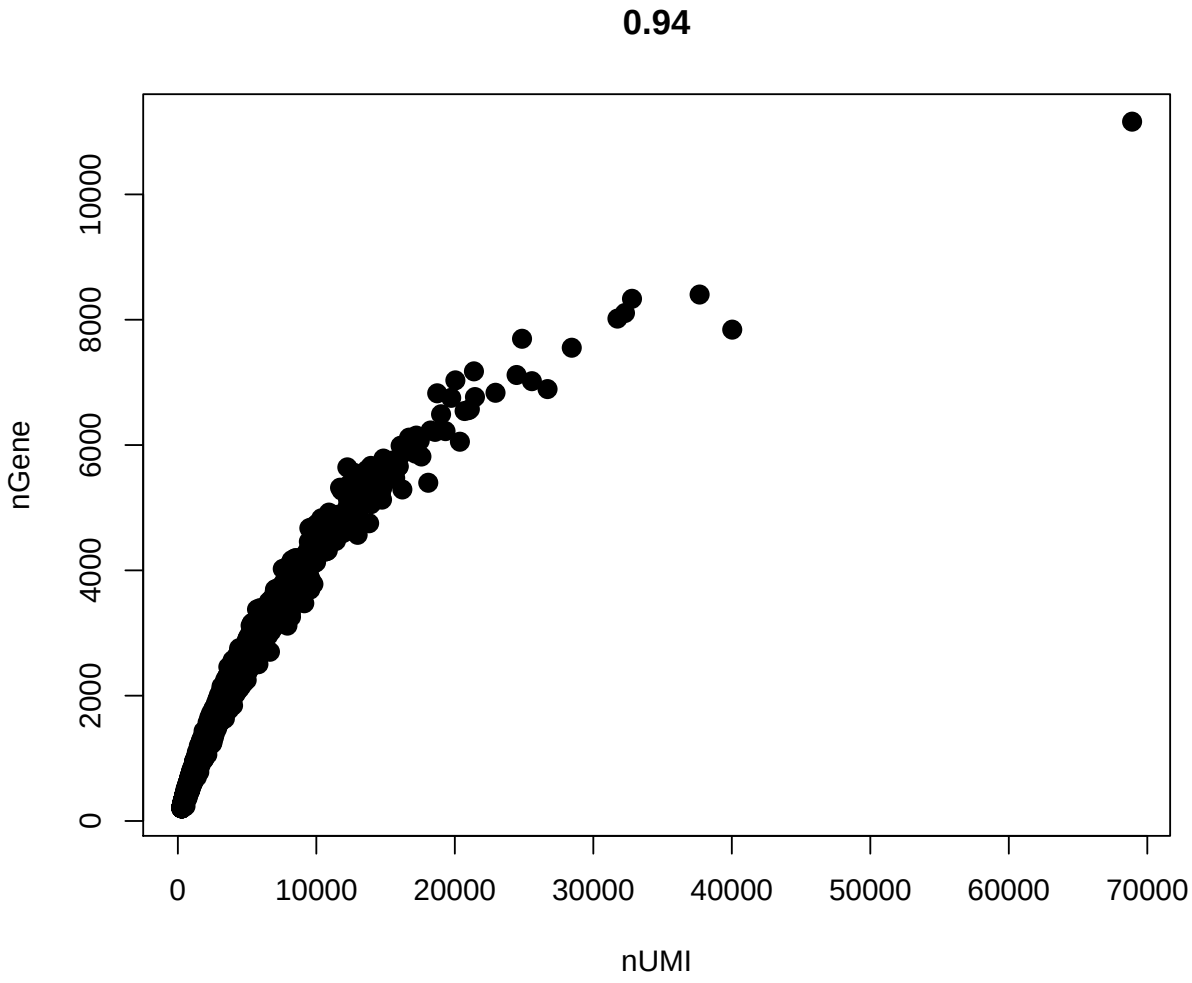
Violin plot:

```
Seurat::VlnPlot(object = mouse$sObj, features.plot = c("nGene", "nUMI"),
                nCol = 2, group.by = "orig.ident", y.log = TRUE)
```



Gene Plot:

```
Seurat::GenePlot(object = mouse$sObj, gene1 = "nUMI", gene2 = "nGene")
```



2.7.2 Regress Out

To quote authors of the paper: "To reduce the effects of library quality and complexity on cluster identity, a linear model was used to regress out effects of the number of transcripts and genes detected per nucleus (using the 'RegressOut' function in the Seurat software package)."

To quote Seurat: "[T]he `RegressOut` function has been deprecated, and replaced with the vars.to.regress argument in `ScaleData`."

In brevity, we attempt to "regress-out" uninteresting sources of variation , including technical noise, batch effects, cell-cycle stage. This is done by learning a linear model to predict gene expression based on user-defined variables.

```
mouse$sObj <- Seurat::ScaleData(mouse$sObj, vars.to.regress = c('nGene', 'nUMI'),
                                do.scale = FALSE, do.center = FALSE)
```

This is time consuming for our big dataset. Alternatively, we may load a precomputed object:

```
load_path <- file.path(basepath, 'data/MouseObj_scaled.RData')
load(load_path)
```

2.7.3 Find variable genes

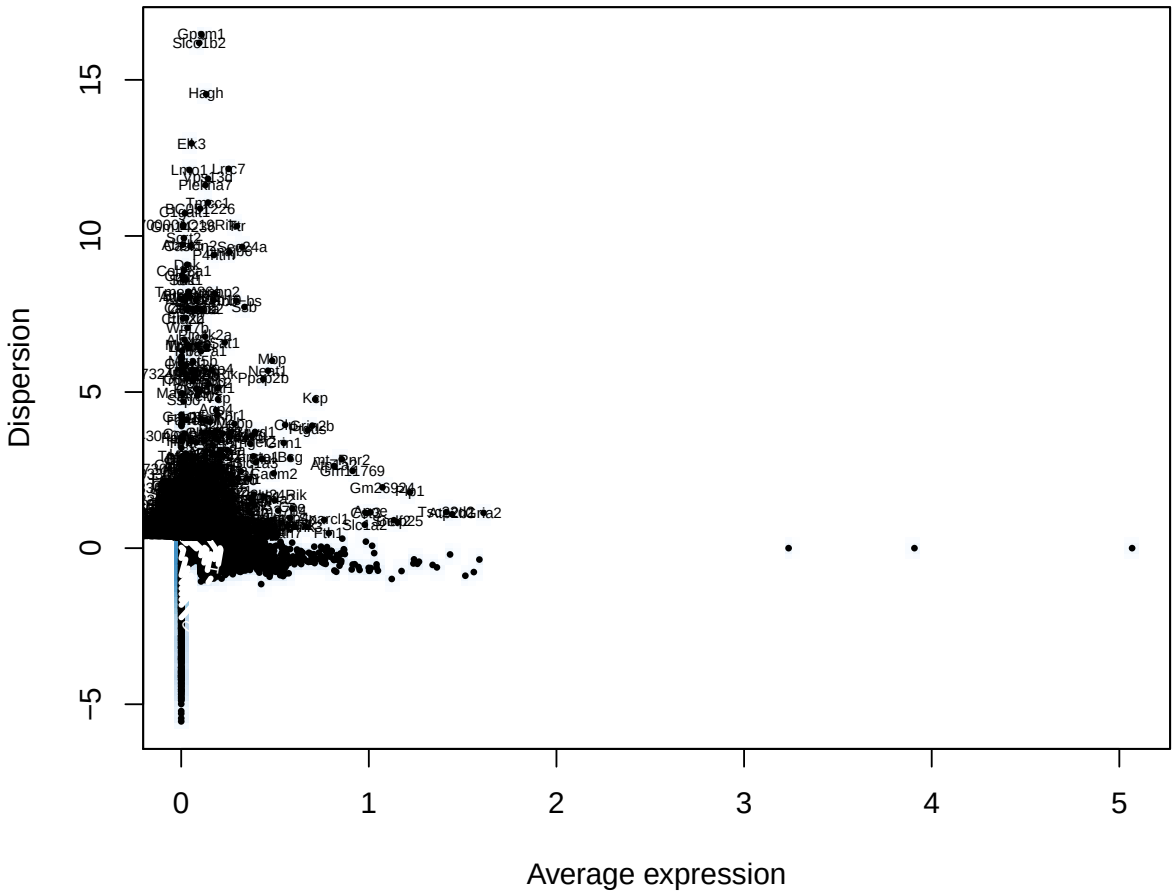
Quoting: "To select highly variable genes, we fit a relationship between mean counts and coefficient of variation using a gamma distribution on the data from all of the genes and ranked genes by the extent of excess variation as a function of their mean expression (using a threshold of at least 0.2 difference in the coefficient of variation between the empirical and the expected and a minimal mean transcript count of 0.005)."

The above discription unfortunately does not allow us to unambigiously identify the exact procedure for discriminating highly variable genes. However, we note that Seurat features a function that can be used for this purpose. We set the paremetrs mirroring the ones used in the paper and also based on other available analyses.

Note that LogVMR is logarithm of variance to mean ratio, which is similar to the logarithm of coefficient of variation and note also that we define the thresholds in non-log space.

```
mouse$sObj <- Seurat::FindVariableGenes(mouse$sObj, mean.function = Seurat::ExpMean,
                                        dispersion.function = Seurat::LogVMR,
                                        x.low.cutoff = 0.005,  y.cutoff = sqrt(0.2),
                                        num.bin = 50)
```

```
## Warning in KernSmooth::bkde2D(x, bandwidth = bandwidth, gridsize = nbin, :
## Binning grid too coarse for current (small) bandwidth: consider increasing
## 'gridsize'
```



2.7.4 Dimensionality reduction - PCA

Next, we reduce the dimensionality of the data with PCA. The dimensionality of the reduced representation is arbitrary but should be sufficient to identify significant PCs in the following step.

Quoting the paper: "We used a DGE matrix consisting only of variable genes as defined above, scaled and log-transformed, and then reduced its dimensions with PCA."

```
mouse$sObj <- Seurat::RunPCA(mouse$sObj, pc.genes = mouse$sObj@var.genes,
                             pcs.compute = 20, genes.print = 5)
```

```
## [1] "PC1"
## [1] "Atp2b1" "Celf2"  "Gria2"  "Snap25" "Grin2b"
## [1] ""
## [1] "Apoe"    "Cst3"    "Atp1a2"  "Slc1a2"  "Sparcl1"
## [1] ""
## [1] ""
## [1] "PC2"
## [1] "Plp1"  "Bsg"   "Rgs5"  "Flt1"  "Itm2a"
## [1] ""
```

```
## [1] "Gria2"  "Slc1a2" "Gm3764" "Ttyh1"  "Clu"
## [1] ""
## [1] ""
## [1] "PC3"
## [1] "Bsg"   "Rgs5"  "Flt1"  "Itm2a" "Ly6c1"
## [1] ""
## [1] "Plp1"  "Mag"   "Mbp"   "Neat1" "Mobp"
## [1] ""
## [1] ""
## [1] "PC4"
## [1] "Tsc22d2" "Ptgds"   "Slc1a2"  "Kcp"     "Atp1a2"
## [1] ""
## [1] "Gm11769" "Fth1"    "Gm26924" "Atp2b1"  "Ppfia2"
## [1] ""
## [1] ""
## [1] "PC5"
## [1] "Atp2b1"  "Gria2"   "Tsc22d2" "Celf2"   "Fth1"
## [1] ""
## [1] "Snap25" "Gad2"   "Sept7"  "Gad1"   "Fut9"
## [1] ""
## [1] ""
```

```
Seurat::PrintPCAParams(mouse$sObj)
```

```
## Parameters used in latest PCA calculation run on: 2017-12-31 14:46:50
## =============================================================================
## PCs computed    Genes used in calculation    PCs Scaled by Variance Explained
##     20                  2432                              TRUE
## -----------------------------------------------------------------------------
## rev.pca
##  FALSE
## -----------------------------------------------------------------------------
## Full gene list can be accessed using
##  GetCalcParam(object = object, calculation = "RunPCA", parameter = "pc.genes")
```

Quoting: "We [...] chose the most significant principal components (or PCs) based on the largest eigen value gap ...".

This can be done by looking for PCs that have high enrichment at low p-values (dashed line represents uniform distribution).

```
mouse$sObj <- Seurat::JackStraw(mouse$sObj, num.pc = 20,
                                num.replicate = 100, do.print = TRUE)
```

This is time consuming for our big dataset. Alternatively, we may load a precomputed object:

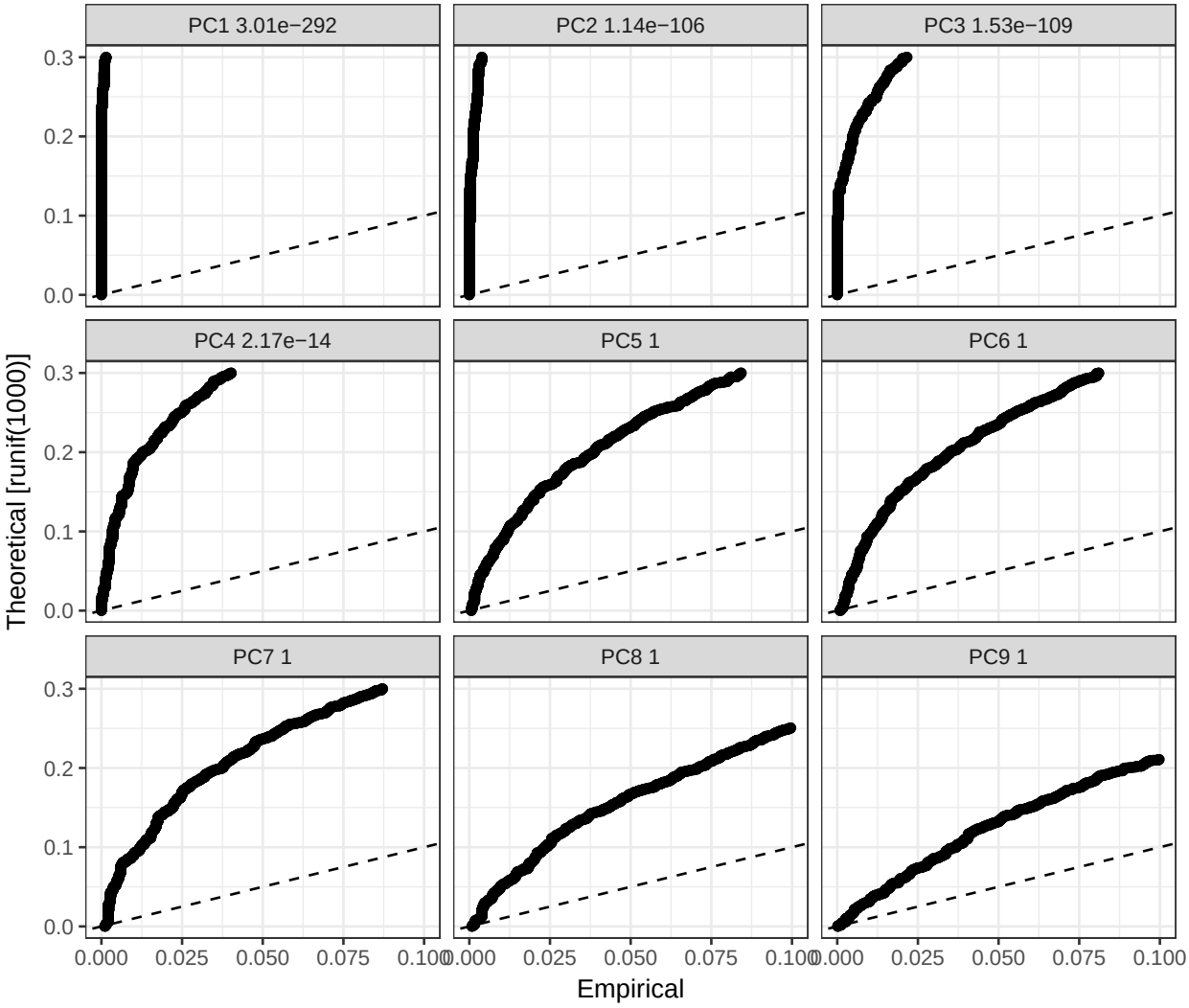
```
mouseObj_path <- file.path(basepath, 'data/mouseObj_jackstraw_tsne.RData')
load(mouseObj_path)
Seurat::PrintPCAParams(mouse$sObj)
```

```
## Parameters used in latest PCA calculation run on: 2017-12-29 07:24:19
## =============================================================================
## PCs computed    Genes used in calculation    PCs Scaled by Variance Explained
##     20                  4985                              TRUE
## -----------------------------------------------------------------------------
## rev.pca
##  FALSE
## -----------------------------------------------------------------------------
```

```
## Full gene list can be accessed using
##  GetCalcParam(object = object, calculation = "RunPCA", parameter = "pc.genes")
```

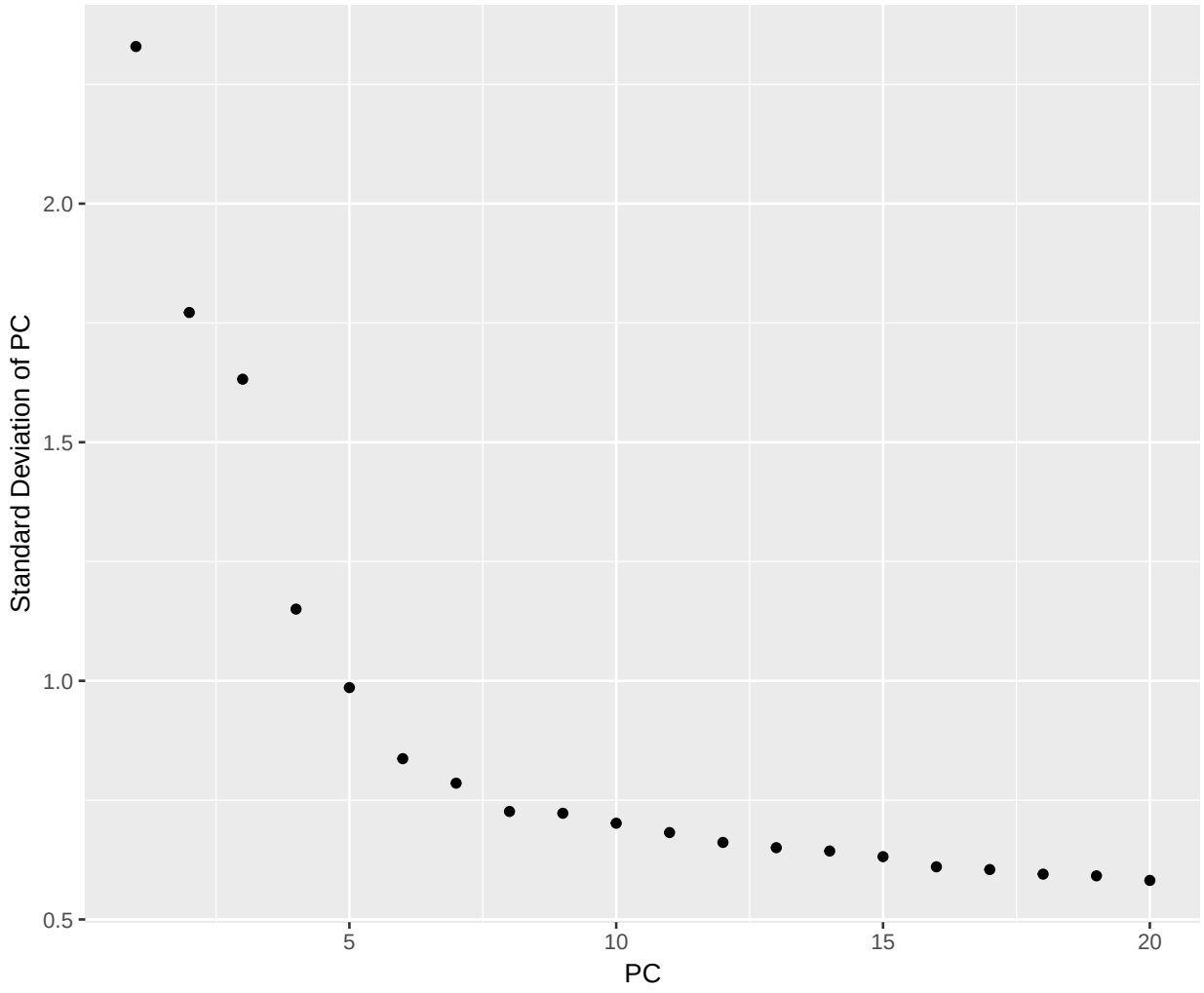
```
Seurat::JackStrawPlot(mouse$sObj, PCs = 1: 9, nCol = 3)
```

```
## Warning: Removed 32092 rows containing missing values (geom_point).
```



We can also use more approximate technique and look at the explained variance plot:

```
Seurat::PCElbowPlot(mouse$sObj)
```



From the plots, it looks like 7 may be appropriate number of PCs to retain.

2.7.5 Dimensionality Reduction - tSNE

Next we compute 2D embedding using tSNE.

Quoting: "We generated a 2D nonlinear embedding of the nuclei profiles using tSNE. The scores along the top significant PCs estimated above were used as input to the algorithm ([...] with a maximum of 2,000 iteration [..] and setting the perplexity parameter to 100.)"

```
mouse$sObj <- Seurat::RunTSNE(mouse$sObj, reduction.use = "pca", dims.use = 1:7,
                              dim.embed = 2, perplexity = 100, max_iter = 2000)

# mouse$tsne <- Rtsne::Rtsne((mouse$sObj@dr$pca@cell.embeddings[ ,grep("PC[1-7]$",
#                              colnames(mouse$sObj@dr$pca@cell.embeddings))]),
#                            pca = FALSE, perplexity = 100, max_iter = 2000, dims = 2)
Seurat::PrintTSNEParams(mouse$sObj)
```

This is time consuming for our big dataset. Alternatively, we may use the previously precomputed result:

```
mouseObj_path <- file.path(basepath, 'data/mouseObj_jackstraw_tsne.RData')
```

```
#load(mouseObj_path)
Seurat::PrintTSNEParams(mouse$sObj)
```

```
## Parameters used in latest TSNE calculation run on: 2017-12-30 08:36:41
## =============================================================================
## Reduction use          do.fast          dim.embed
##      pca                TRUE                2
## -----------------------------------------------------------------------------
## Dims used in calculation
## =============================================================================
## 1 2 3 4 5 6 7
```

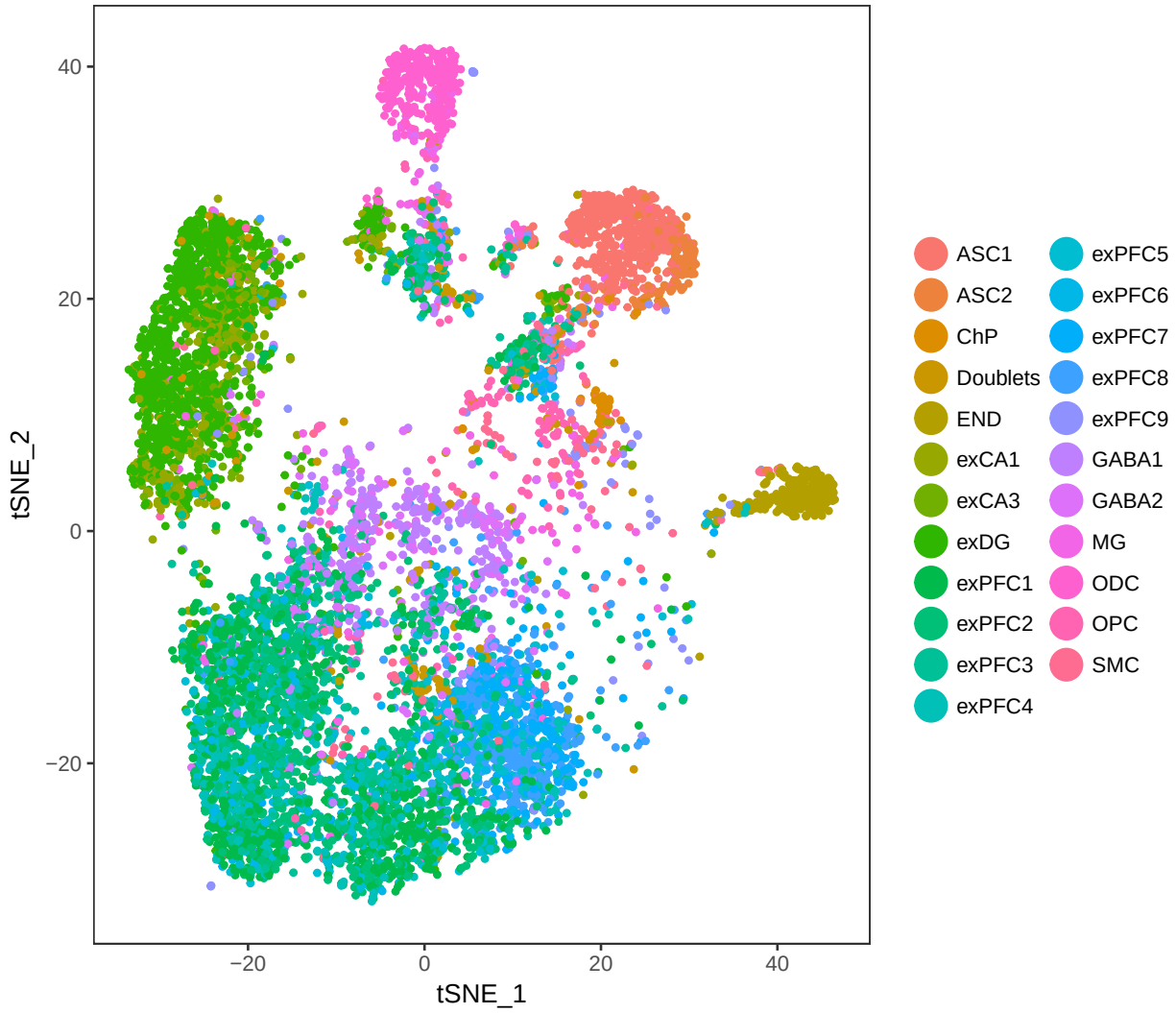
2.7.6 Visualization and comparison

Finally, we plot the data in the space of 2D tSNE embedding. We also color the cells by the cell-type assignment as reported in the data made available with the publication. This assignment was obtained by graph-based clustering algorithm. Briefly, the cells were embedded in K-nearest neighbor (KNN) graph based on the euclidean distance in PCA space, with edges drawn between cells with similar gene expression patterns. The algorithm partitioned this graph into highly interconnected 'quasi-cliques' or 'communities'.

```
mouse.clusterfile <- file.path(basepath, 'data/Mouse_Meta_Data_with_cluster.txt')
mouse$clusters <- read.table(mouse.clusterfile , sep = "\t", header = TRUE)
mouse$clusters <- mouse$clusters[-1, ]
mouse$clusters$NAME <- gsub("_", ".", mouse$clusters$NAME)
mouse$clusters$NAME <- gsub("-", "_", mouse$clusters$NAME)
unassigned <- grepl("Unclassified[0-9]", mouse$clusters$Cluster)
mouse$clusters$Cluster[unassigned] <- "Unclassified1"
mouse$clusters$ClusterID[unassigned] <-
  min(as.integer(unique(mouse$clusters$ClusterID[grepl("Unclassified[0-9]",
                                                       mouse$clusters$Cluster)])))

mouse$clusters <- mouse$clusters[match(rownames(mouse$sObj@meta.data),
                                       mouse$clusters$NAME), ]
levels(mouse$clusters$Cluster) <- c(levels(mouse$clusters$Cluster), "unkn.")
mouse$clusters$Cluster[is.na(mouse$clusters$ClusterID)] <- "unkn."
new_ident <- setNames(mouse$clusters$Cluster, names(mouse$sObj@ident))
mouse$sObj@ident <- new_ident
```

```
Seurat::TSNEPlot(mouse$sObj,
                 cells.use = mouse$sObj@cell.names[!is.na(mouse$clusters$ClusterID) &
                                                     mouse$sObj@ident != "Unclassified1"],
                 do.label = FALSE)
```



Overall, we see that the cells represented by our data tend to form clusters according to the cell-type assignment obtained in the publication. This is however not generally true for all cells.

3 Conclusion

In this study we have analyzed DroNc-seq data from the paper that pioneered the method. The steps of bioinformatic analysis are mostly identical to those for single-cell Drop-seq. Specifically, we have:

- Preprocessed the raw data,
- aligned reads to reference genome,
- filtered the reads and cells,
- summarized the transcript abundances in digital expression matrix,
- identified highly variable genes,
- reduced dimensionality of data with PCA,
- identified significant PCs based on the largest eigenvalue gap,

- visualized the data in 2D-space after nonlinear tSNE embedding,
- and visually confirmed cell-type assignments.

Overall, we can say that we have arrived to qualitatively similar results as the authors of the publication. The reason why the results are not identical is mainly the fact that the data-analysis pipeline comprises multitude of tunable parameters that are often interchangeable or, conversely, mutually exclusive and usually neither well documented nor clearly identified in the publication. The necessity to work with big data and to run significant part of the analysis on cluster complicated our pursuit as it prolonged time of a development cycle, with wall-time of individual scripts exceeding 12 hours in extreme cases. Given the time available for the project, this limited possible number of iterations we could conduct to search the parameter space for our analysis. Furthermore, some combination of parameters produced a DGE that was too big to be worked with locally and therefore were not investigated further.

For future work on the project, we recommend thoroughly investigating the wrapping of DropSeq Tools by dropSeqPipe as the latter, although conveniently chaining the commands and aiming for rerpoducibility, may obviate some settings of the former and may introduce spurious errors to the analysis when applied to slightly different problem then initially developed for. Next, multiple settings for extraction of DGE matrix should be tested out as these allow us to filter genes/cells/reads/UMIs based on several criteria and may therefore have important impact for downstream analysis. Similarly, Seurat functions for filtering of genes/cells and identification of variable genes can be tested with different parameters to observe impact on the following dimensionality reduction. Furthermore, clustering can be performed with `Seurat::FindClusters()` function, while also varying the number of PCs retained as input, and the result quantitatively compared to class assignment obtained in the discussed publication.

Acknowledgment

We thank authors of Drop-seq Tools, dropSeqPipe for the developed and published code. Despite its imperfections, our pursuit would be impossible without the extensive work that went into development of this software. Similarly, we are grateful for computing resources available to students at ETH via Euler. Last but not least, thanks go to the Prof. Robinson and his students and co-workers for organizing the STA426 course that this work was done in scope of. Although very time consuming, the course provided ample opporunites to gain skills in the field of statistical analysis of high-throughput omics data.

Appendix

Listing of commands useful when working on cluster

```
# change execution time of a job
bmod -W 24:00 JOBID
# list available queues
bqueues
# switch job to another queue
bswitch QUEUENAME JOBID
# move file between storages
mv $HOME/file.ext $SCRATCH/file.ext
# ensure java has NUMCORES available
export _JAVA_OPTIONS="-XX:ParallelGCThreads=NUMCORES"
# get help for some Drop-seq Tool
/path/to/DropSeqScript -- -h
```

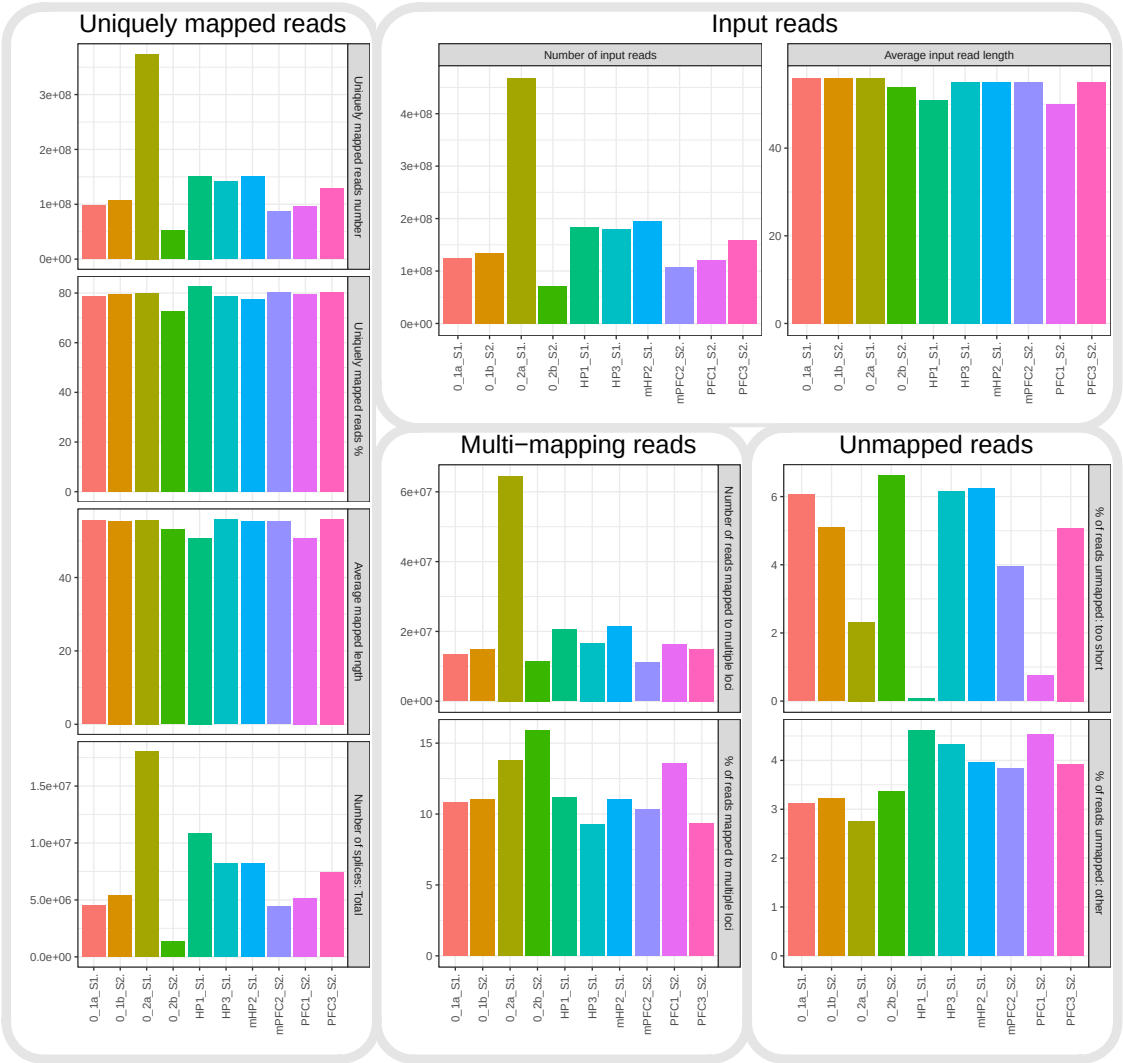


Figure 6: STAR Alignement Summary

STAR Summary

Figure 6.

MultiQC Summary

MultiQC summary for all files is available in document `data\multiqc_report.html`.

Cluster Job Reports

Example of cluster job reports is available in `data\cluster_reports`.

Sessioninfo

```
sessionInfo()
```

```
## R version 3.4.1 (2017-06-30)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 15063)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] Biobase_2.36.2      BiocGenerics_0.22.1 Matrix_1.2-10
##
## loaded via a namespace (and not attached):
##   [1] Seurat_2.1.0         diffusionMap_1.1-0   Rtsne_0.13
##   [4] VGAM_1.0-4           colorspace_1.3-2     ggridges_0.4.1
##   [7] class_7.3-14         modeltools_0.2-21    mclust_5.4
##  [10] rprojroot_1.2        htmlTable_1.11.1     base64enc_0.1-3
##  [13] proxy_0.4-20         rstudioapi_0.7       DRR_0.0.2
##  [16] flexmix_2.3-14       lubridate_1.7.1      prodlim_1.6.1
##  [19] mvtnorm_1.0-6        ranger_0.8.0         codetools_0.2-15
##  [22] splines_3.4.1        R.methodsS3_1.7.1    mnormt_1.5-5
##  [25] doParallel_1.0.11    robustbase_0.92-8    knitr_1.18
##  [28] tclust_1.3-1         RcppRoll_0.2.2       Formula_1.2-2
##  [31] caret_6.0-78         ica_1.0-1            broom_0.4.3
##  [34] gridBase_0.4-7       ddalpha_1.3.1        cluster_2.0.6
##  [37] kernlab_0.9-25       R.oo_1.21.0          sfsmisc_1.1-1
##  [40] compiler_3.4.1       backports_1.1.1      assertthat_0.2.0
##  [43] lazyeval_0.2.0       lars_1.2             acepack_1.4.1
##  [46] htmltools_0.3.6      tools_3.4.1          bindrcpp_0.2
##  [49] igraph_1.1.2         gtable_0.2.0         glue_1.2.0
##  [52] reshape2_1.4.2       dplyr_0.7.4          Rcpp_0.12.13
##  [55] NMF_0.20.6           trimcluster_0.1-2    gdata_2.18.0
##  [58] ape_5.0              nlme_3.1-131         iterators_1.0.9
##  [61] fpc_2.1-10           psych_1.7.8          timeDate_3042.101
##  [64] gower_0.1.2          stringr_1.2.0        irlba_2.3.1
##  [67] rngtools_1.2.4       gtools_3.5.0         DEoptimR_1.0-8
##  [70] MASS_7.3-47          scales_0.5.0         ipred_0.9-6
##  [73] RColorBrewer_1.1-2   yaml_2.1.14          pbapply_1.3-3
##  [76] gridExtra_2.3        ggplot2_2.2.1        pkgmaker_0.22
##  [79] segmented_0.5-3.0    rpart_4.1-11         latticeExtra_0.6-28
##  [82] stringi_1.1.5        foreach_1.4.4        checkmate_1.8.5
##  [85] caTools_1.17.1       ggjoy_0.4.0          lava_1.5.1
```

```
##  [88] dtw_1.18-1           SDMTools_1.1-221     rlang_0.1.2
##  [91] pkgconfig_2.0.1      prabclus_2.2-6       bitops_1.0-6
##  [94] evaluate_0.10.1      lattice_0.20-35      ROCR_1.0-5
##  [97] purrr_0.2.4          bindr_0.1            labeling_0.3
## [100] recipes_0.1.1        htmlwidgets_0.9      cowplot_0.9.2
## [103] tidyselect_0.2.3     CVST_0.2-1           plyr_1.8.4
## [106] magrittr_1.5         R6_2.2.2             gplots_3.0.1
## [109] Hmisc_4.0-3          dimRed_0.1.0         sn_1.5-1
## [112] withr_2.1.1          foreign_0.8-69       mixtools_1.1.0
## [115] survival_2.41-3      scatterplot3d_0.3-40 nnet_7.3-12
## [118] tsne_0.1-3           tibble_1.3.4         KernSmooth_2.23-15
## [121] rmarkdown_1.6        grid_3.4.1           data.table_1.10.4-3
## [124] FNN_1.1              ModelMetrics_1.1.0   digest_0.6.12
## [127] diptest_0.75-7       xtable_1.8-2         numDeriv_2016.8-1
## [130] tidyr_0.7.2          R.utils_2.6.0        stats4_3.4.1
## [133] munsell_0.4.3        registry_0.5
```

References

Naomi Habib, Yinqing Li, Matthias Heidenreich, Lukasz Swiech, Inbal Avraham-Davidi, John J Trombetta, Cynthia Hession, Feng Zhang, and Aviv Regev. Div-Seq: Single-nucleus RNA-Seq reveals dynamics of rare adult newborn neurons. *Science*, 353(6302):925–928, 2016. ISSN 0036-8075. doi: 10.1126/science.aad7038. URL http://science.sciencemag.org/content/353/6302/925.

Naomi Habib, Inbal Avraham-Davidi, Anindita Basu, Tyler Burks, Karthik Shekhar, Matan Hofree, Sourav R Choudhury, François Aguet, Ellen Gelfand, Kristin Ardlie, David A Weitz, Orit Rozenblatt-Rosen, Feng Zhang, and Aviv Regev. Massively parallel single-nucleus RNA-seq with DroNc-seq. *Nature Methods*, 14:955, aug 2017. URL http://dx.doi.org/10.1038/nmeth.4407http://10.0.4.14/nmeth.4407https://www.nature.com/articles/nmeth.4407{#}supplementary-information.

Benjamin Lacar, Sara B Linker, Baptiste N Jaeger, Suguna Rani Krishnaswami, Jerika J Barron, Martijn J E Kelder, Sarah L Parylak, Apuã C M Paquola, Pratap Venepally, Mark Novotny, Carolyn O'Connor, Conor Fitzpatrick, Jennifer A Erwin, Jonathan Y Hsu, David Husband, Michael J McConnell, Roger Lasken, and Fred H Gage. Nuclear RNA-seq of single neurons reveals molecular signatures of activation. *Nature Communications*, 7:11022, apr 2016. URL http://dx.doi.org/10.1038/ncomms11022http://10.0.4.14/ncomms11022https://www.nature.com/articles/ncomms11022{#}supplementary-information.

Blue B Lake, Rizi Ai, Gwendolyn E Kaeser, Neeraj S Salathia, Yun C Yung, Rui Liu, Andre Wildberg, Derek Gao, Ho-Lim Fung, Song Chen, Raakhee Vijayaraghavan, Julian Wong, Allison Chen, Xiaoyan Sheng, Fiona Kaper, Richard Shen, Mostafa Ronaghi, Jian-Bing Fan, Wei Wang, Jerold Chun, and Kun Zhang. Neuronal subtypes and diversity revealed by single-nucleus RNA sequencing of the human brain. *Science*, 352(6293):1586–1590, 2016. ISSN 0036-8075. doi: 10.1126/science.aaf1204. URL http://science.sciencemag.org/content/352/6293/1586.

Evan Z. Macosko, Anindita Basu, Rahul Satija, James Nemesh, Karthik Shekhar, Melissa Goldman, Itay Tirosh, Allison R. Bialas, Nolan Kamitaki, Emily M. Martersteck, John J. Trombetta, David A. Weitz, Joshua R. Sanes, Alex K. Shalek, Aviv Regev, and Steven A. McCarroll. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell*, 161(5):1202–1214, dec 2015. ISSN 10974172. doi: 10.1016/j.cell.2015.05.002. URL http://dx.doi.org/10.1016/j.cell.2015.05.002.

Karthik Shekhar, Sylvain W Lapan, Irene E Whitney, Nicholas M Tran, Evan Z Macosko, Monika Kowalczyk, Xian Adiconis, Joshua Z Levin, James Nemesh, Melissa Goldman, Steven A McCarroll, Constance L Cepko, Aviv Regev, and Joshua R Sanes. Comprehensive Classification of Retinal Bipolar Neurons by Single-Cell Transcriptomics. *Cell*, 166(5):1308 – 1323.e30, 2016. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2016.07.054. URL http://www.sciencedirect.com/science/article/pii/S0092867416310078.

Christoph Ziegenhain, Beate Vieth, Swati Parekh, Björn Reinius, Amy Guillaumet-Adkins, Martha Smets, Heinrich Leonhardt, Holger Heyn, Ines Hellmann, and Wolfgang Enard. Comparative Analysis of Single-Cell RNA Sequencing Methods. *Molecular Cell*, 65(4):631 – 643.e4, 2017. ISSN 1097-2765. doi: https://doi.org/10.1016/j.molcel.2017.01.023. URL http://www.sciencedirect.com/science/article/pii/S1097276517300497.