Published in final edited form as:

Methods Mol Biol. 2019 ; 1919: 145–159. doi:10.1007/978-1-4939-9007-8_11.

Bioinformatics Analysis of Single-Cell RNA-Seq Raw Data from iPSC-Derived Neural Stem Cells

Jeffrey Kim and Marcel M. Daadi

Abstract

This chapter describes a pipeline for basic bioinformatics analysis of single-cell sequencing data (see Chap. 10: Single-Cell Library Preparation). Starting with raw sequencing data, we describe how to quality check samples, to create an index from a reference genome, to align the sequences to an index, and to quantify transcript abundances. The curated data sets will enable differential expression analysis, population analysis, and pathway analysis.

Keywords

Single cell; Neural stem cells; Fluidigm; C1; Bioinformatics; Single-cell analysis; Single-cell RNA-seq; RNA-seq; Singular; SCDE; Kallisto; Sleuth; DAVID; BBDuk; FASTQC

1 Introduction

The general rule of thumb for RNA-seq analysis is to first quality check the raw sequences and remove those of low quality. Then the sequences are aligned to an index built from a reference genome. After the quantification of transcript abundances, differential expression may be observed and expression pathways may be analyzed. RNA-seq analysis can be performed as genome guided or de novo. De novo analysis is performed without a reference genome due to unknown or incomplete references. This protocol utilizes a human reference genome and therefore will follow a genome-guided pipeline. Alignment of reads for RNA-seq covers noncontinuous portions of reference genomes that result from transcript splicing. Therefore, alignment tools are designed to find optimal alignment of noncontinuous sequences. Herein, we utilize a light-weight pipeline that can be used on a standard desktop computer. This pipeline is designed to be relatively simple for a beginner in the field of bioinformatics. Subsequent analysis described in this pipeline was written with Mac OS X and under the assumption that paired-end sequencing data is being used. Lines starting with “\$” will be typed into the terminal or shell, while lines starting with “>” will be used in R. When designating the location of a file, /path/ to/ notation will be used. Please note the provided text /path/to/ is not the actual path to be used which is entirely dependent on the physical location within a user’s hard drive.

2 Materials

This section will cover the tools and their installation, necessary for single-cell analysis pipeline.

2.1 Computer Specifications

When performing bioinformatics analysis, a Unix-based operating system, such as Linux or Mac OS X, is recommended. Unix allows for the use of command-line-based tools that can be automated using custom scripts. Command-line tools can be accessed through a shell. The shell will be used to install and use command-line tools that perform index building, alignment, and quantification of raw sequencing data. The following protocols will be utilizing shell on Mac OS X. Also, consider the computer strength since index building and alignment require a lot of processing power. Certain processes can be sped up by partitioning more CPU cores to a certain task. Another limiting factor when analyzing is computer storage space. As sequencing data creates very large files, it is easy to run out of hard drive space. Space may be managed as needed depending on the amount of data generated. If using a Windows PC, there are ways to utilize command-line tools. However, for ease of access for beginners in bioinformatics, a Unix-based computer with adequate storage is preferable.

2.2 RStudio

RStudio is an open source software that uses the R framework. Its graphical interface allows for easy usage of R. However, some features in the R packages discussed in this protocol do not work with RStudio and therefore must be performed through normal R. To install RStudio, first install R, which can be found in the following website: <https://cran.r-project.org/>. Download R for the relevant platform. Once installed, proceed to download RStudio from the following website: <https://www.rstudio.com/products/rstudio/download/>.

2.3 FASTQC

When first obtaining raw sequencing data, check the quality of the sequencing experiment. Installation of this program does not require the use of command line. This program requires Java 7 or higher to run. Website to download FASTQC: <https://www.bioinformatics.babraham.ac.uk/projects/fastqc/>.

2.4 BBDuk

BBDuk is a trimming tool that is part of the BBTools toolset. BBTools has a variety of other bioinformatics tools; however, here only BBDuk is used. Download it from sourceforge.net/projects/bbmap/. Move the downloaded file to a working directory of choice.

To install, go to the terminal or shell and change to the working directory and then input the following command:

```
$ tar --xvzf BBMap_(version).tar.gz
```

This will create a subfolder named bbmap, which contains the scripts and files needed to use BBDuk.

2.5 Kallisto

Kallisto is a program for quantifying transcript abundances through pseudoalignment for rapid determination of compatibility of reads. Kallisto is also used to build an index from a reference genome.

```
To install kallisto, use the following commands in shell:
$ ruby --e "$(curl --fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
$ brew tap homebrew/science
$ brew install kallisto
```

2.6 Sleuth

Sleuth is an R package designed for usage downstream of Kallisto.

```
To install Sleuth use the following packages in R:
> source("http://bioconductor.org/biocLite.R")
> biocLite("rhd5")
> install.packages("devtools")
> devtools::install_github("pachterlab/sleuth")
```

2.7 Singular

Singular is an analysis toolset offered gratis through Fluidigm. Its ease of use through a graphical interface makes it easy for beginners in bioinformatics to perform basic analysis. However, this R package does not work with Mac computers. Singular analysis toolset software and practice sets can be downloaded from www.fluidigm.com/software. To install the package, in R, go to the packages tab and click install packages from local files. Then, select the downloaded zipped file.

```
Once installed, in R, type:
> Library(fluidigmSC)
> firstRun()
```

2.8 SCDE

SCDE is an R package used in the statistical analysis of single-cell RNA-seq data. Use this to observe differential expression across samples.

```
To install SCDE use the following packages in R:
```

```
> source("https://bioconductor.org/biocLite.R")
> biocLite("scode")
```

2.9 DAVID

To use DAVID, proceed to the following URL: david.ncifcrf.gov.

3 Methods

3.1 QC and Trimming of Raw Sequences

1. Load fastq files into FASTQC.
2. Observe per base sequence quality. If the curve dips below a Phred score of 28, the sequence will require trimming (*see* Note 1). If more than half of the sequence is below 28, file may not be adequate for downstream analysis and should be removed. Any adapter sequences present in the overrepresented sequences will also need to be removed.
3. Perform the following command in terminal or shell. For the inputs, direct the pathway to the location of files. For pairedend sequencing data, provide two inputs, one for each sample. The outputs will be designated with a new directory. The qtrim parameter designates the direction of the trim (either left or right). In the following example, right is selected with "r." The trimq parameter designates the Phred score threshold where trimming should occur. Here we set the threshold to 28, which will trim the bases to the right of where the Phred score dips below 28.

Perform the following command in terminal or shell to trim sequence:

```
$ bbduk.sh in1=/path/to/read1.fq in2=/path/to/read2.fq out1=/newpath/to/clean1.fq out2=/newpath/to/clean2.fq qtrim=r trimq=28
```

4. Repeat this process with all pairs that fail the initial QC.
5. Repeat QC for trimmed sequences to verify the effectiveness of trimming. These sequences will be used downstream.

3.2 Index Building

1. Perform the following command in the terminal or shell. The generated index file will be output into the current directory. Change directory prior to this command.

To generate an index:

```
$ kallisto index --i nameOfChoice /path/to/cDNA.FASTA
```

¹-Phred quality score is the measurement of the quality of raw sequencing data. A Phred score dictates the probability of an incorrect base call. A score of 30 means that 1 in 1000 base calls is incorrect, while a score of 20 means 1 in 100 is incorrect.

3.3 Alignment and Quantification

1. Perform the following command after building the index. Here we set the number of bootstraps to 100. This can be changed if desired. The input files are the trimmed fastq inputted as pairs.

To perform pseudoalignment:

```
$ kallisto quant --i /path/to/nameOfIndex -b 100 -o nameOfDirectoryOutput /  
path/to/input1_1.fastq / path/to/input1_2.fastq
```

2. To expedite the process of quantifying the samples, write a script to automate the process. Create a plain text file that lists the name of the fastq files. Then, create a plain text file with the following commands.

Create a plain text file with the following:

```
#!/bin/sh  
while read i;do  
echo$i  
kallisto quant -i /path/to/kallisto_index -b 100 -o / path/to/output_dir/$  
{i} /path/to/files_dir/${i} 1_1.fastq /path/to/files_dir/${i}1_2.fastq  
done < filenames.txt
```

Name this file `runkallisto.sh`. To run the script, type the following in terminal or shell.

To run the script, type into terminal or shell:

```
$ . /runkallisto.sh
```

3. Alternatively, use the following command to perform the same function as “quant” but without bootstrap. First make a plain text file called “batch.txt,” which includes columns for #id, file1, and file2 names.

Alternative method for pseudoalignment:

```
$ kallisto pseudo -i kallisto_index -o output -b batch.txt
```

3.4 Filtering Transcript Abundances and Annotation with Sleuth

1. To begin, run the following commands to open necessary R packages.

Load the following packages:

```
> library("sleuth")  
> library("biomaRt")
```

2. The following commands will designate the pathway to the location of the kallisto files. Use .h5 files as they contain boot-straps. First designate the base directory.

Set base directory:

```
> base_dir <- "/Users/username/workingDirectory"
```

3. Command to designate the directory containing abundance.h5 files.

Designate abundance.h5 file directory:

```
> sample_id <- dir(file.path(base_dir, "name_of_folder"))
```

4. The following command will designate the files.

Designate bundance.h5 files:

```
> kal_dirs <- sapply(sample_id, function(id) file.path(base_dir, "name_of_folder", id, "abundance.h5"))
```

5. Create a .txt file that lists the name of the samples and their condition. First column is labeled "Sample_id," and the second column is "Condition." This will be used for the following command.

Define s2c variable:

```
> s2c <- read.table(file.path(base_dir, "sample_id.txt"), header = TRUE, stringsAsFactors = FALSE)
```

6. The following commands will now designate the file location to the sample of interest.

Designate file location:

```
> s2c <- dplyr::select(s2c, sample=Sample_id, Condition)
```

7. The following command sets the path for the s2c variable to the kal_dirs directory.

Set path to kal_dirs:

```
> s2c <- dplyr::mutate(s2c, path=kal_dirs)
```

8. Next, use biomaRt to annotate transcript ids using the Ensembl database. Designate use of the Ensembl database for human genes.

Designate biomaRt to use Ensembl database with human genes:

```
> mart <- useMart("ensembl", "hsapiens_gene_ensembl")
```

9. Designate desired kind of output from annotation. Here transcript id, gene id, and gene name are desired.

Define t2g variable:

```
> t2g <- getBM(c("ensembl_transcript_id", "ensembl_gene_id",  
"external_gene_name"), mart =mart)
```

10. The following command will rename selected parameters.

Rename parameters of t2g variable:

```
> t2g <- dplyr::rename(t2g, target_id = ensembl_transcript_id, ens_gene =  
ensembl_gene_id, ext_gene = external_gene_name)
```

11. Perform the following command for transcript-level analysis including gene names.

Perform transcript-level analysis:

```
> so <- sleuth_prep(s2c, ~Condition, target_mapping = t2g,  
aggregation_column = ens_gene)  
> so <- sleuth_fit(so)  
> so <- sleuth_fit(so, ~1, reduced)  
> so <- sleuth_lrt(so, reduced, full)
```

12. The following command brings up a new window that displays filtered data.

Live display:

```
> sleuth_live(so)
```

13. Go to the summaries tab and to kallisto table to view the transcript abundances. Save this table as it will be needed to generate a data matrix. Processed data can also be viewed for alignment QC metrics.

3.5 Analysis with Singular

1. Before analysis of filtered data can begin, transcript abundances must be compiled into a plain text matrix. To do this, simply use an excel spreadsheet and save as a txt file. The rows designate genes, while the columns designate sample names.

2. Begin by attaching the package into R (*see* Note 2).

Load fluidigmSC package:

```
> library(fluidigmSC)
```

3. When performing the following command, a new command prompt will appear. Input the data matrix and sample sheet. The sample data sheet created in the previous section can be used. This command will detect outliers in the data set.

Detect outliers:

```
> identifyOutliers()
```

4. Use generated outliers.fso for analysis. Type the following:

Perform auto analysis:

```
> autoAnalysis()
```

Another command prompt will appear. This time, input outliers.fso and sample sheet. For genes of interest, choose defined in the expression file to designate all genes; otherwise, designate which genes or how many of the top variable genes will be represented. Then choose the destination for the output. auto-Analysis will display a PCA plot, tSNE plot, a hierarchical clustering heatmap, and violin plots.

5. Enter command to select custom colors and symbols:

Select colors and symbols:

```
> fldm_exp <- setSampleGroupColorAndSymbols (fldm_exp)
```

6. Enter command to display a 3D PCA.

Display 3D PCA:

```
> display3DPCAScore(pca = fldm_pca, x_axis=1, y_axis=2, z_axis=3,
locate=TRUE)
```

7. Enter command to create an “anova object” which will be used to generate a volcano plot.

Create an anova object:

```
> anova <- ANOVA (fldm_exp)
```

².Analysis with Singular will be done through normal R as certain functions may not properly work in RStudio.

8. Enter command to provide a list of genes that are significantly different between two sample groups.

Generate a list of significantly different genes:

```
> anova_gene_list <- getTopANOVAGenes (anova, top_gene_num=200,
pvalue_threshold = 0.05)
```

9. To make a volcano plot, use the following command. CT1 and CT2 are placeholder names for the sample groups. Fold-change threshold can be changed to a value of preference. The default fold-change threshold is set to one, which in actuality equates to a two-fold change.

Create a volcano plot:

```
> volcano_gene_list <- foldChangeAnalysis(anova, sample_group1 = "CT1",
sample_group2 = "CT2", foldchange_threshold = 1, pvalue_threshold = 0.05,
display_plot = TRUE, locate = TRUE)
```

10. Enter command to save the data.

To save data:

```
> saveData()
```

3.6 Analysis with SCDE: Differential Expression

Begin by attaching the necessary packages.

Load the packages:

```
> library(scde)
> library(org.Hs.eg.db)
```

1. Assign a counts variable to the data matrix.

Define counts variable:

```
> counts <- read.delim("/path/to/data.table.txt", row.names = "gene_id")
```

2. Enter command to designate the condition groups or cell types.

Designate conditions:

```
> sg <- factor(gsub("(one|two).*", "\\1", colnames (counts)),
levels=c("one", "two"))
```

```
> names(sg) <- colnames(counts)
> table(sg)
```

3. The following command is used to work around a possible error.

Error fix:

```
> counts<-apply(counts,2,function(x) {storage.mode(x) <- integer; x})
```

4. The following command will fit the data table to an error model. The `min.size.entries` variable may be changed to fit the data set. The number of CPU cores may also be changed to preference.

Fit to error model:

```
>o.ifm<-scde.error.models(counts = counts, groups = sg, n.cores = 2,
threshold.segmentation = TRUE, save.crossfit.plots = FALSE, save.model.plots
= FALSE, min.size.entries = 500, verbose = 1)
```

5. The following command filters out cells that do not show positive correlation with the expected expression magnitudes.

Filter cells:

```
> valid.cells <- o.ifm$corr.a > 0
> table(valid.cells)
> o.ifm <- o.ifm[valid.cells,]
```

6. The following command defines a grid of expression magnitude values on which the numerical calculations will be carried out. Here a grid of 400 points is used.

Define grid of expression magnitude values:

```
> o.prior <- scde.expression.prior(models = o.ifm, counts = counts,
length.out = 400, show.plot = FALSE)
```

7. Enter command to define a factor which two groups of cells are to be compared, which are the rows of the data set.

Define groups:

```
> groups <- factor(gsub("(one|two).*", "\\1", rownames(o.ifm)),
levels=c("one", "two"))
> names(groups) <- row.names(o.ifm)
```

8. Enter command to run differential expression test on all genes.

Perform differential expression test:

```
> ediff <- scde.expression.difference(o.ifm, counts, o.prior, groups =
groups, n.randomizations = 100, n.cores = 2, verbose = 1)
```

9. Command to display top upregulated genes; “head command”.

Display top upregulated genes:

```
> head(ediff[order(ediff$Z, decreasing = TRUE),])
```

10. Command to display top downregulated genes; “tail command”.

Display top downregulated genes:

```
> tail(ediff[order(ediff$Z, decreasing = TRUE),])
```

11. Command to create a table with the results.

Create a results table:

```
> write.table(ediff[order(abs(ediff$Z), decreasing = TRUE),], file =
"results.txt", row.names = TRUE, col.names = TRUE, sep = "\t", quote =
FALSE)
```

12. Command provides a web browser application where differentially expressed genes can be browsed (see Note 3).

Browse differentially expressed genes:

```
> scde.browse.diffexp(ediff, o.ifm, counts, o.prior, groups = groups, name
= "diffexpl", port = 1299)
```

13. Command to allow the observation of differential expression in a single gene of interest.

View a single gene or interest:

```
> scde.test.gene.expression.difference ("GENEname", models = o.ifm, counts =
counts, prior = o.prior)
```

3.7 Analysis with SCDE: Pathway and Gene Set Overdispersion Analysis

1. Load the count matrix.

³The command does not work under RStudio and will crash the program. Instead use base R.

Load counts matrix:

```
> counts <- read.delim("/path/to/data.table.txt", row.names = "gene_id")
```

2. The following command filters out poor cells and genes.

Filter cells:

```
> cd <- clean.counts(counts)
```

3. The following commands are used to work around an error.

Error fix:

```
> counts<-apply(counts,2,function(x) {storage.mode(x) <- integer; x})
> cd <- counts
> x <- gsub("^Hi_(.*)_", "\\1", colnames(cd))
```

4. The following command fits the data set to an error model. Change the minimum library size based on preference. The number of CPU cores can also be assigned.

Fit data to error model:

```
> knn <- knn.error.models(cd, k=ncol(cd)/4, n.cores = 2, min.count.threshold
= 2, min.nonfailed = 5, max.model.plots = 10, min.size.entries = 572)
```

5. The following normalizes variance and generates a variance plot.

Normalize variance:

```
> varinfo <- pagoda.varnorm(knn, counts=cd, trim=3/ ncol(cd), max.adj.var. =
3, n.cores = 2, plot=TRUE)
```

6. Display genes with high adjusted variance.

Display genes with high adjusted variance:

```
> sort(varinfo$arv, decreasing=TRUE)[1:10]
```

7. The following command controls gene coverage and sequencing depth.

Control for gene coverage and sequencing depth:

```
> varinfo <- pagoda.subtract.aspect(varinfo, colSums(cd[, rownames(knn)]>0))
```

8. Translate gene names to ids.

Translate gene names to ids:

```
> ids <- unlist(lapply(mget(rownames(cd), org.Hs.egALIAS2EG, ifnotfound =
NA), function(x) x[1]))
> rids <- names(ids); names(rids) <- ids
```

- 9.** The following command assigns gene ontologies of interest from ids to gene names.

Assign gene ontologies:

```
> gos.interest <- unique(c(ls(org.Hs.egGO2ALLEGS) [1:100],
"GO:NumbersOfInterest"))
> go.env <- lapply(mget(gos.interest, org.Hs.egGO2ALLEGS), function(x)
as.character(na.omit (rids[x])))
```

- 10.** Remove gene ontologies with too few or too many genes.

Filter gene ontologies:

```
> go.env <- clean.gos(go.env)
> go.env <- list2env(go.env)
```

- 11.** Calculate weighted first principal component magnitudes for each gene set.

Calculate first principal component:

```
> pwpca <- pagoda.pathway.wPCA(varinfo, go.env, n. components = 1, n.cores =
6, n.internal.shuffles = 50)
```

- 12.** Show PC1 variance magnitude as a function of set size.

Show PC1 variance magnitude:

```
> df <- pagoda.top.aspects(pwpca, return.table = TRUE, plot = TRUE, z.score
= 1.96)
```

- 13.** The following procedure will determine “de novo” gene clusters in the data and build a background model for the expectation of the gene cluster-weighted principal component magnitudes.

Determine gene clusters:

```
> clpca <- pagoda.gene.clusters(varinfo, trim=7.1/ ncol(varinfo$mat),
n.clusters = 50, n.cores = 6, plot=TRUE)
```

- 14.** The set of top aspects can be recalculated taking de novo gene clusters into account.

Recalculate top aspects:

```
> df <- pagoda.top.aspects(pwpca, clpca, return.table = TRUE, plot = TRUE,
z.score = 1.96)
```

- 15.** Obtain information on significant aspects of transcriptional heterogeneity.

Top aspects of transcriptional heterogeneity:

```
> tam <- pagoda.top.aspects(pwpca, clpca, n.cells = NULL, z.score =
qnorm(0.05, lower.tail = FALSE))
> hc <- pagoda.cluster.cells(tam, varinfo)
```

- 16.** Combine pathways that are driven by the same set of genes.

Combine pathways based on same gene sets:

```
> tamr <- pagoda.reduce.loading.redundancy(tam, pwpca, clpca)
```

- 17.** Combine aspects that show similar patterns.

Combine aspects:

```
> tamr2 <- pagoda.reduce.redundancy(tamr, distance.threshold = 0.9, plot =
TRUE, cell.clustering = hc, labRow = NA, labCol = colnames(cd), box = TRUE,
margins = c(20, 5), trim = 0)
```

- 18.** View top aspects clustered by pattern similarity.

View top aspects:

```
> col.cols <- rbind(groups = cutree(hc, 2))
> pagoda.view.aspects(tamr2, cell.clustering = hc, box = TRUE, labCol = NA,
margins = c(0.5, 20), col.cols = NULL)
```

- 19.** Open the pagoda app to allow for interactive browsing and exploration of output (see Note 4).

Open pagoda app:

```
> app <- make.pagoda.app(tamr2, tam, varinfo, go.env, pwpca, clpca,
```

⁴The command does not work in RStudio and must be performed with base R.

```
col.cols = col.cols, cell.clustering = hc, title = "Title")
> show.app(app, "TITLE", browse=TRUE, port=1468)
```

20. View the pathway according to gene ontology gene set of interest.

```
View pathways of interest:
> pagoda.show.pathways(c("GO: NumbersOfInterest"), varinfo, go.env, cell.
clustering = hc, margins = c(1,5), show.cell.dendrogram = TRUE,
showRowLabels = TRUE, showPC = TRUE)
```

3.8 DAVID Analysis

1. On the DAVID website, click "Functional Annotation" in the side bar.
2. Provide DAVID with a list of genes of interest.
3. Select an identifier of interest. In this case use `ensembl_gene_id`, `ensembl_transcript_id`, or `official_gene_symbol`.
4. Select the list type as gene list.
5. Submit list.
6. Select Homo sapiens for species, or other relevant species.
7. Click annotation clustering to view enriched pathways.
8. To save the information, click download file. Then, copy and paste the information into an excel spreadsheet.

Acknowledgment

The authors thank the members of the Daadi laboratory for their helpful support and suggestions. This work was supported by the Worth Family Fund, the Perry & Ruby Stevens Charitable Foundation and the Robert J., Jr. and Helen C. Kleberg Foundation, the NIH primate center base grant (Office of Research Infrastructure Programs/OD P51 OD011133), and the National Center for Advancing Translational Sciences, National Institutes of Health, through Grant UL1 TR001120.