RAS: AN END TO END SUITE FOR SINGLE-CELL AND STANDARD RNA-SEQ DATA ANALYSIS

BY

NIKHIL KUMAR

A thesis submitted to the

Graduate School—New Brunswick

Rutgers, The State University of New Jersey

and

The Graduate School of Biomedical Sciences

in partial fulfillment of the requirements

For the degree of

Master of Science

Graduate Program in Biomedical Engineering

Written under the direction of

Dr. Li Cai

And approved by

New Brunswick, New Jersey

May, 2017

ABSTRACT OF THE THESIS

RAS: AN END TO END SUITE FOR SINGLE-CELL AND STANDARD RNA-SEQ DATA ANALYSIS

by NIKHIL KUMAR

Thesis Director:

Dr. Li Cai

As RNA sequencing (RNA-Seq) becomes more affordable to process, the number of sequencing data has increased at a rapid rate. A major challenge in RNA-Seq experiments is the analysis of large amount of data generated by next-generation sequencing. RNA-Seq data analysis usually requires advanced Linux/Unix experience for software installation and tedious commands. This required skill set hinders scientists who are interested in analyzing RNA-Seq data but do not have Linux/Unix experience. Here, we describe an end to end pipeline for RNA-Seq Data Analysis Suite (RAS), which streamlines and speeds up analysis time for both standard and single-cell and standard RNA-Seq data. In addition, RAS is a standalone web application and contains an interface where Linux/Unix skills are not required for the data analysis. RAS creates a web server, where the user can interact with the pipeline on a browser. In addition, Since since RAS utilizes distributed computing, it uses computational resources efficiently. RAS The pipeline was validated on single-cell RNA-Seq datasets publicly available at the Gene Expression Omnibus (GEO) from the NCBI.

Acknowledgements

I would like to thank:

- My advisor, Dr. Li Cai, and for his support on this project. Your valuable feedback in our lab meetings is greatly appreciated.
- The members of Dr. Cai's lab for creating a pleasant and helpful environment.
- The professors under whom I have conducted research during my 5 years at Rutgers. Your dedication to mentorship has helped me sharpen my skills as a scientist.
- My friends for making college such an enjoyable experience.
- The students at the Rutgers C.A.V.E. (Collaborative Academic Versatile Environment) for helping me develop my skills in Linux
- The organizers of HackRU, you created a fun and nurturing environment which has instilled within me a passion for software engineering. The free pizza was also nice.
- My professors at Rutgers, the Biomedical Engineering department, and the Graduate School for harboring my academic growth.
- The members of my committee for dedicating their time to be part of my defense.

Dedication

I would like to dedicate this thesis to:

- My parents for providing a nurturing environment, instilling strong values and showing me the path of self-motivation and dedication.
- My great-uncle for invoking within me a curiosity and interest in science from my early childhood.
- My brother for your shameless puns and creating a fun-filled atmosphere.

Table of Contents

List of Figures

Chapter 1

Introduction

Transcriptome analysis allows us to analyze and quantity RNA species in a biological sample at a specific moment in time. RNA-Seq has revolutionized such analysis. Standard RNA-seq analysis measures the mean value from thousands to millions of cells, thus, the signal of individual cells was ignored. However, most of the biological systems contain heterogeneity of cell population, transcriptome profiling from individual cells is crucial for the identification and characterization of different cell types. Recently, a new technique known as single-cell RNA-Seq has been developed to provide a higher resolution of cellular differences and a better understanding of the function of an individual cell. In the recent years, a wealth of bioinformatics tools have been developed to tackle the informatics challenges for mapping and analysis of both standard and single-cell RNA-Seq data. The adaption of these tools often requires Linux/Unix and programming skills. With the steep learning curve for Linux/Unix, this required skill set hinders scientists who are interested in analyzing RNA-Seq data but do not have Linux/Unix experience [1]. In addition, many bioinformatic tools do not have a graphical user interface and requires tedious usage of the Linux/Unix command line.

RNA-seq data analysis suite (RAS) is a pipeline aimed at providing an end to end data analysis of single-cell and standard RNA-Seq experiments by utilizing powerful openly-available bioinformatic tools. RAS is a standalone web application and contains an interface where Linux/Unix skills are not required for the data analysis. RAS creates a web server, where the user can interact with the pipeline on a browser. The pipeline is computationally efficient and powerful using concepts from distributed computing and artificial intelligence while at the same being user-friendly and requiring no Linux/Unix or programming experience.

To validate the effectiveness and accuracy of RAS, we ran several studies on neural stem cells (NSCs) using single-cell sequencing data openly available from the Gene Expression Omnibus (GEO) at the national center for biotechnology and information (NCBI). It was believed that the brain's poor regenerative abilities can be attributed to its lack of neural stem cells. However, the discovery of stem cells in the adult brain has created a new area of research on regenerative treatment for brain disorders [2]. This combined with the new single-cell RNA-Seq technology has allowed scientists to isolate and analyze dynamic changes in the NSC state. Studies have demonstrated that NSCs exist in mainly two brain areas, the subgranular zone (SGZ) of the hippocampus and the subventricular zone (SVZ) of the forebrain. NSCs can be classified into two main sub-groups: quiescent (qNSCs) and active (aNSCs) [3][4]. In this work, we develop RAS and analyze single-cell RNA-seq datasets of NSCs from the SVZ and SGZ. The results from RAS validate the usefulness of RAS and provide new insights into the molecular mechanisms of transition between quiescent to active states and determine the molecular differences of NSCs in the SVZ and the SGZ.

Chapter 2

Methods

The pipeline is constructed primarily using the programming language Python [5]. A module known as Flask [6] is used to create a server, in which the user can interact with the program using a web browser. Single-cell RNA-Seq data of NSCs from the SVZ (NCBI GEO accession #GSE67833) [5] and SGZ (NCBI GEO accession #GSE71485) [6] were used for validation and further studies.

For an RNA-Seq experiment a dynamic form was created in Javascript [7] for the user to input his/her files or url links of sra files for RNA-Seq analysis. This allowed user to add or delete conditions (e.g., cells from the SVZ or cells from the SGZ) and add or delete replicates for each condition with ease. In addition, regular expressions are used to ensure the user is giving the appropriate information. The data from the form is then downloaded (e.g., from GEO)/uploaded (user's own data) into a folder named with a unique experiment id and a well-organized directory.

RAS maintains a specific directory structure for each experiment. This is shown in Figure 2.1. Each replicate has its own folder to store the log files from data preparation programs. The users name and email is also stored in each of his or her experiment folders. This gives the administrator a person to contact when he or she needs to delete experiments to free up disk space.

A quality check function for the sequencing data has been implemented for the pipeline with the intuition to easily interact with data sources at the GEO. The user can supply an accession number (e.g., GSE67833 and GSE71485) and the pipeline will retrieve all the sra files from the GEO database. The pipeline will then decompress the files using Fastq-dump and analyze them using the program FastQC [8][9]. The quality check of data is all aggregated into a tab delimited file and emailed back to the user. RAS also allows the

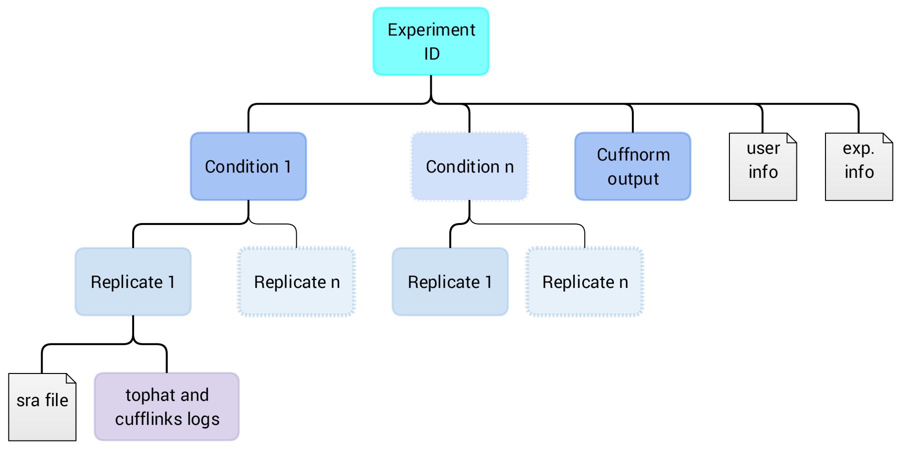


Figure 2.1: Directory structure of RAS. Each replicate of RNA-Seq samples has its own folder to store logs. Each experiment folder contains the users information which is valuable to administrators.

user to visualize the results through a scatter plot. The scatter plot is run on javascript using the ChartJS [10] package. The charts are responsive and whenever a user hovers over a data point a tooltip will show the cell label. The scatter plots of the two acession ids (#GSE67833 and #GSE71485) are shown in Figure 3.9a and 3.9b respectively.

Special care has been made to make sure that the pipeline does not overload the GEO server at NCBI. This includes havening a mandatory wait time after each request. In addition, the pipeline recognizes that the quality check thread and the experiment download thread are running independently. This runs the risk of having two threads downloading from the GEO server at the same time. In order to prevent this, the parallel computing concept of mutual exclusion is used to address this issue. Whenever a thread is downloading data from the GEO it holds a lock that prevents other threads from accessing the GEO. After it is done downloading, it releases the lock, allowing other threads to download.

The architecture diagram of experiment jobs in RAS is shown in Figure 2.2. The experiment queue is held in a MongoDB database to organize multiple experiment jobs. The database allows independent scripts and programs to coordinate together. A python daemon is periodically checking the queue for new jobs. Once a job is found, it gets processed through the pipeline. Once a job is finished, the next job is started automatically without any need for user input.

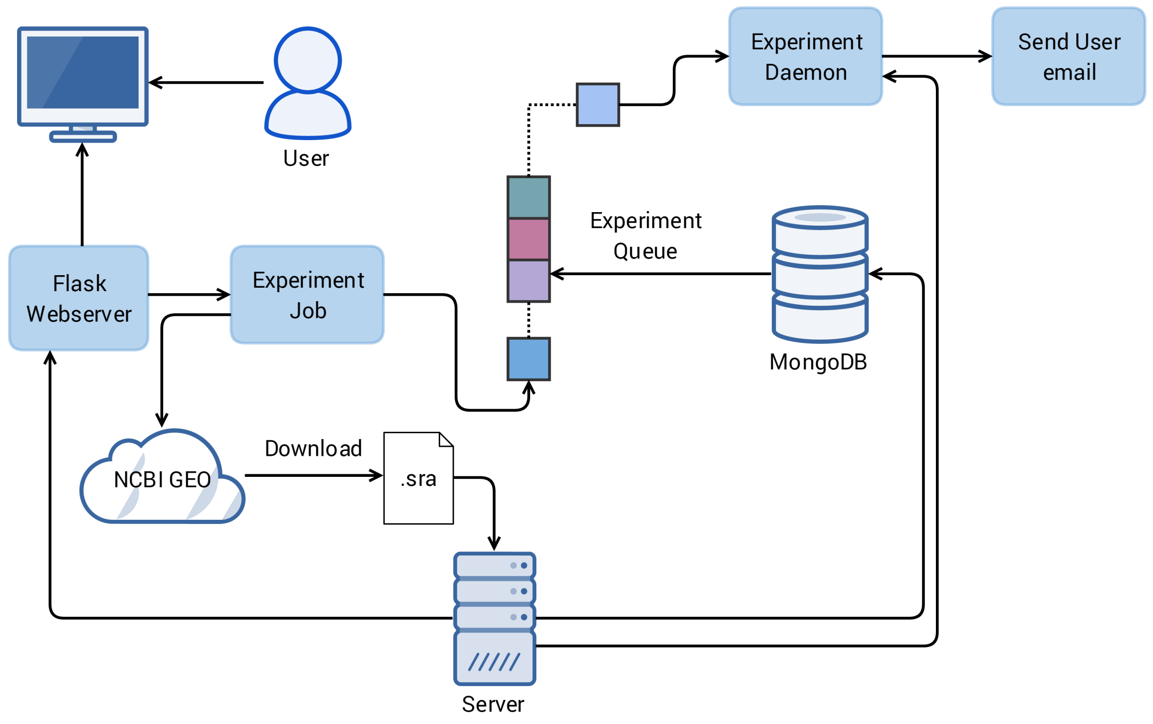


Figure 2.2: Architecture Diagram and Work Flow of RAS. A MongoDB server is used to implement a queue for job handling. The server has many independent components such as the webserver and experiment deamon

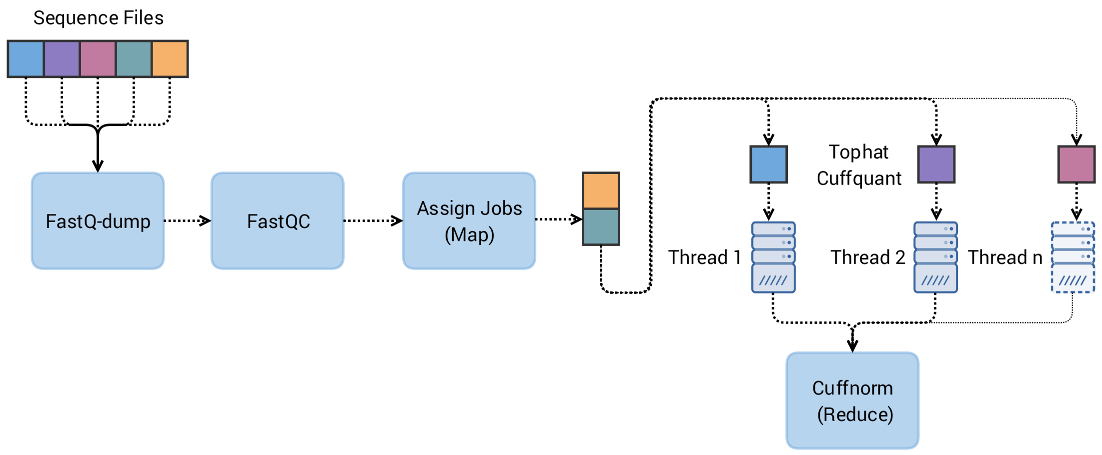


Figure 2.3: RAS Pipeline structure using the MapReduce Model. Multiple threads can be used to process the jobs in parallel.

The experiment is first processed through quality check. Each sequence file is processed through FastQC and the results are stored on the server. Once all the files are finished with the analysis, RAS sends the user an email with links to all the FastQC quality check reports relevant to this experiment. A MapReduce model is then used in the processing of the experiments. The model is shown in Figure 2.3. Each experiment gets divided into numerous blocks where each block is a sequence file. Each block then gets mapped to a pool of user specified threads. This allows sequences to be aligned in parallel running threads. The program TopHat is used for the alignment of the data and further prepared using the cufflinks package [11][12]. The blocks are then combined using cuffnorm (for single-cell RNA-Seq data) or cuffdiff (for standard RNA-Seq data) from the cufflinks package [13]. RAS automatically adjusts for paired-end and single ended sequence data. Once the experiment is finished, the output folder is compressed, stored in the server and a link to the compressed folder is then emailed to the user. Email communication is an important aspect of the pipeline. It allows user to be notified instantly and keeps the user organized. Each email has the unique id on its subject line to prevent confusion.

The pipeline also includes the functionality to aid users in post-alignment analysis. Given two groups/conditions, the pipeline is able to visualize the genes with the largest difference in mean expression using ChartJS (Figure 3.8a and 3.8b). RAS also monitors the status of the server and allows the user to visualize it through responsive graphs. The server status daemon is running in python and checks the server status every minute. It then stores the data on the MongoDB server. The types of information collected are Temperature, Cpu frequency, Network, Disk, Memory, and System Load in Figures 3.9c, 3.9d, 3.9e, 3.9f, 3.9g, and 3.9h respectively. RAS also offers a page where the user see what experiments are on the queue for normalization, differentiation, and quality check analysis.

RAS allows users to use R [14] RNA-Seq analysis packages such as Monocle [15] (for cuffnorm data) and CummeRbund [16] (for cuffidff data) without having any programming or Linux/Unix experience. The user would enter the valid parameters for the particular method he or she wants to be executed. RAS runs an R script with those values by calling the Linux/UNIX system directly. In some cases, the R script is dynamically generated. If the user needs a high resolution figure, a tiff or png format image with 600 dpi can be downloaded. A webpage is returned to the user with the links to the figure. The server disables caching by the browser so that old figures can be removed properly when new ones are generated. This interface greatly simplifies the analysis for the user.

2.1 Single-cell RNA-Seq Analysis

For single-cell RNA-Seq data, RAS has monocle integrated as well as custom methods for analysis. After processing the data the user needs to filter out poor quality cells. A custom script was created where the user can specify a lower bound and an upper bound for the mRNA counts. This allows the user to remove poorly expressed cells or doublet cells, respectively. In addition, RAS can eliminate genes that are not significantly expressed in a certain number of cells where the user can specify the number and significance threshold. After quality control, the user can manually trim out cells. This is ideal for dividing the sample set. To simplify the input, a method is designed to take in an inclusive range designated by a '-' and separate numbers by ','. For example 1,2,4-7 will include cells 1,2,4,5,6, and 7. After quality control, the user can perform a clustering analysis and/or classify the cells using the Monocle package. The classify script works by representing class labels with logical gene value conditions. A special dynamic form is created where the user can create any number of logical statements ($<,\leq,>,\geq$) as well as specify any logical condition across statements (AND, OR). The user can also manually classify the sample using ranges of cell numbers labels. After classification, the user has the options to analyze the data through pseudo-time analysis, trajectory, or differential expression for a specified gene through Monocle.

2.2 Decision Tree

RAS includes a novel application of decisions trees for the differential analysis of single-cells across known groups. The decision tree is an artificial intelligence concept for classification learning [17]. The decision tree finds the best way to organize the cells into their correct groups using their gene expression values. The output is a flowchart where each node represents a gene logic statements (Figure 2.7). The decision tree chooses the genes and cutoff values which will represent the dataset with the fewest number of logic statements. After fitting, the genes in each node represent the significant genes within the dataset. These genes are the best at dividing the data into the specified subgroups. In the Figure 2.8, we see how the decision tree in Figure 2.7 divides SVZ Quiescent and SVZ Active cells. In the first node, the two groups are divided almost entirely with the exception of one cell. In the sample, decision tree we can see that the gene in the first node is much more critical than the node where one SVZ Quiescent cells is separated from the SVZ Active cells. The decision tree is implemented in Python using the Scikit-learn package [18]. This method was extended to perform a more comprehensive analysis.

2.3 Differential Analysis

RAS also includes a differential analysis algorithm which uses the decision tree to obtain significant genes. The decision tree first isolates a specific group from the rest of the cells. The gene in the first node (most significant) is stored and removed from the training gene dataset. The decision tree is then retrained and run again. The process repeats for all the groups until all the groups have a list of significant genes at the user specified length. This isolates clusters of genes which represent the different groups of cells. Using this information and the class of cells in the experiment a heatmap is created where the columns represent each cell and rows represent each gene. The cells are grouped by its cell type and the genes are grouped based on which class of cells they represent. The heatmaps were created in the R package pheatmap [19]. The user has the option to group the data from classification or specify one manually.

2.4 Standard RNA-Seq

For standard RNA-Seq Analysis data, RAS is integrated with the cummeRbund workflow [16]. The first method creates the CuffSet Database and prepares the data for cummeRbund analysis. After that, the user can perform quality control from the FPKM values and/or create a dendogram and analyze the clustered cells. RAS can also generate bar plots for gene expression as well as find similar genes using CummeRbund. After this step, the

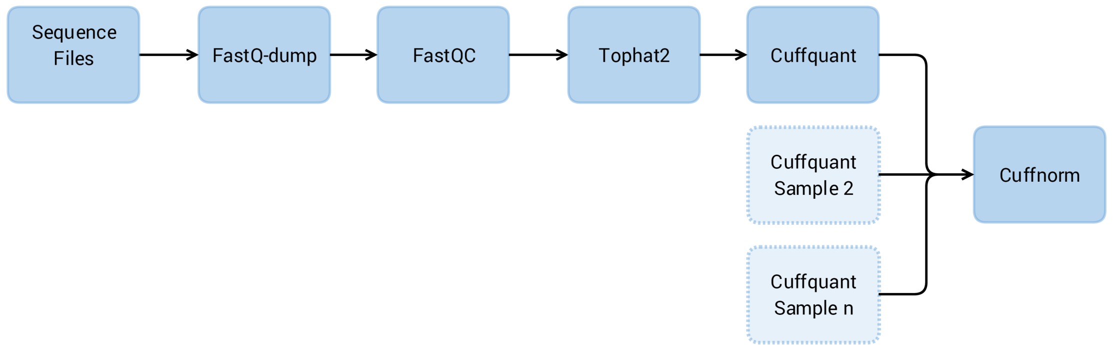


Figure 2.4: Workflow for single-cell RNA-Seq experiments.

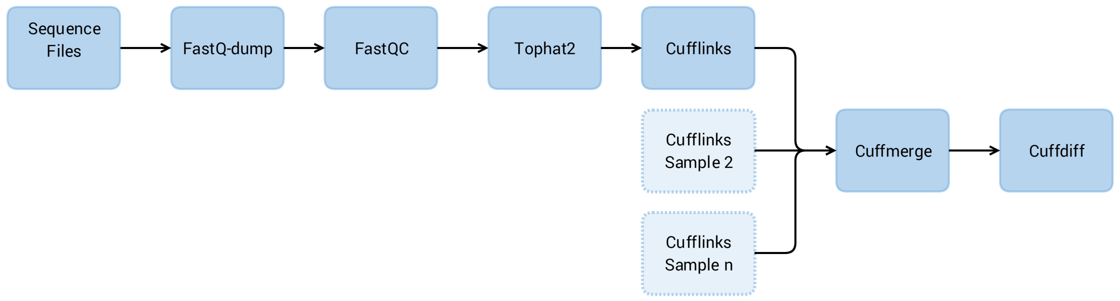


Figure 2.5: Workflow for standard RNA-Seq experiments

user needs to create a significant gene set. RAS allows the user to get significant genes across 2 or all the classes using the appropriate cummeRbund functions. The user can then generate heatmaps and/or k-means clusters of gene expression patterns. For the k-means clustering component, a table is also dynamically created and shown of gene names and its corresponding cluster.

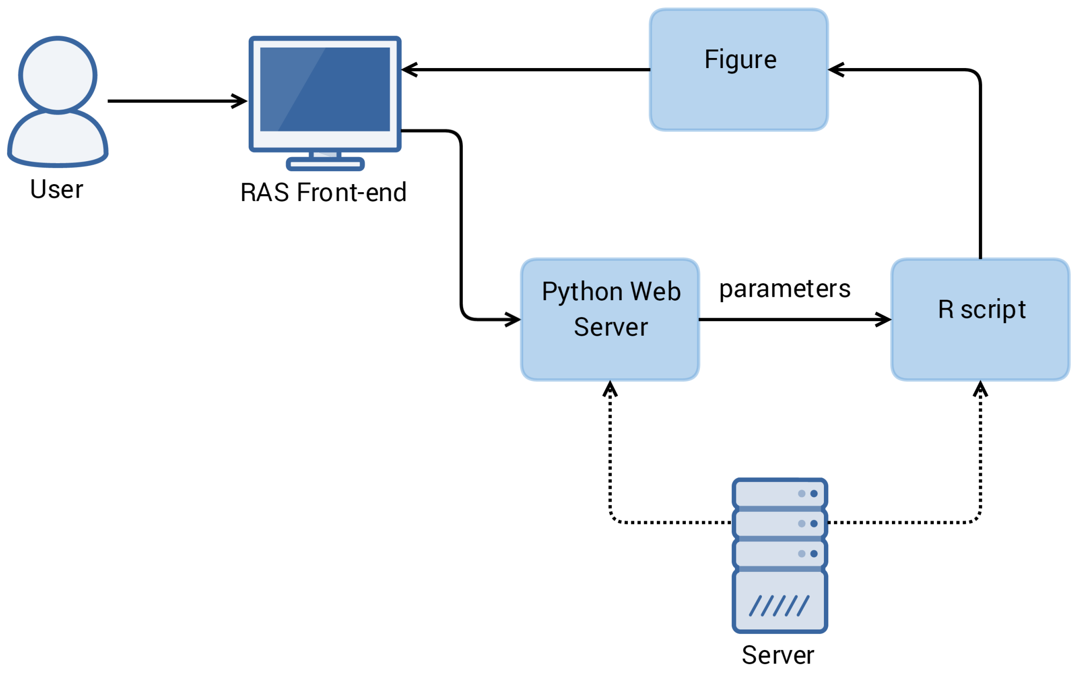


Figure 2.6: Architecture diagram of data analysis in RAS using Python and R

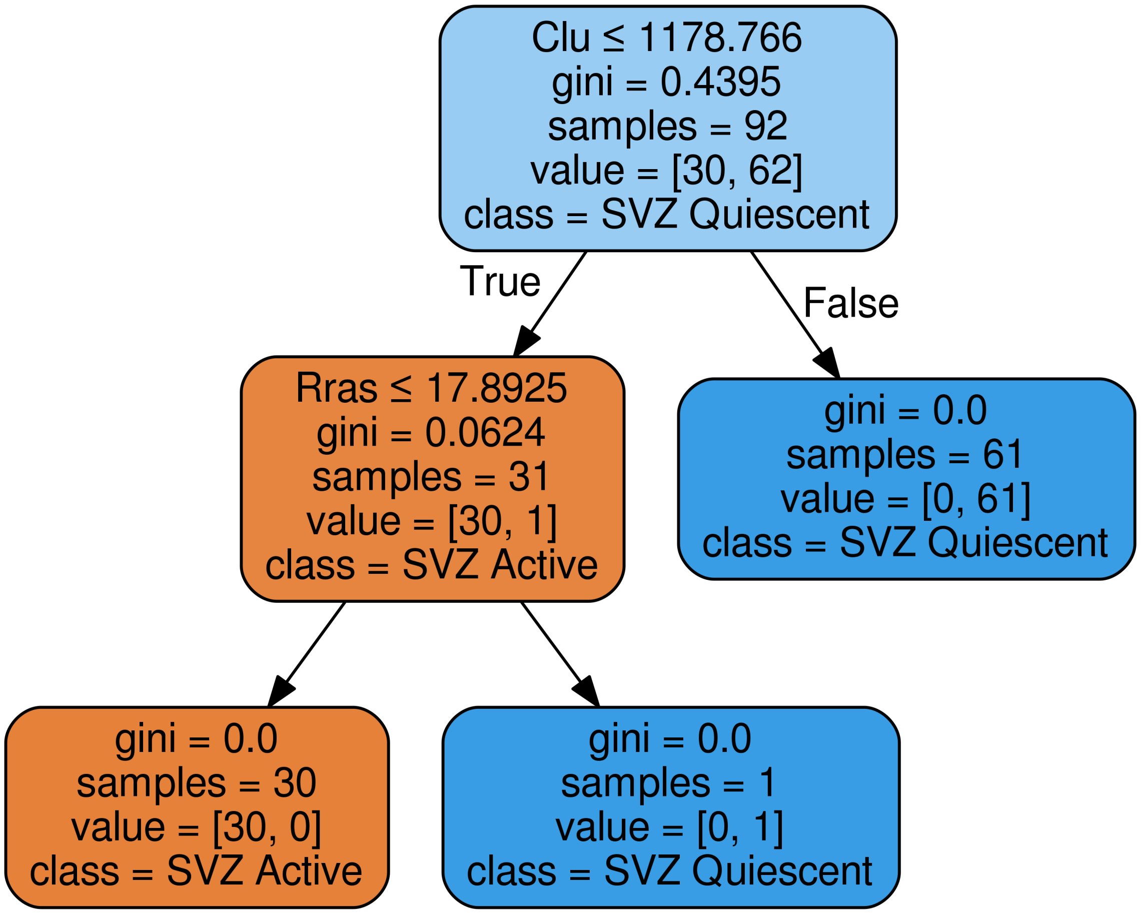


Figure 2.7: Sample Decision Tree of SVZ Quiescent and SVZ Active cells

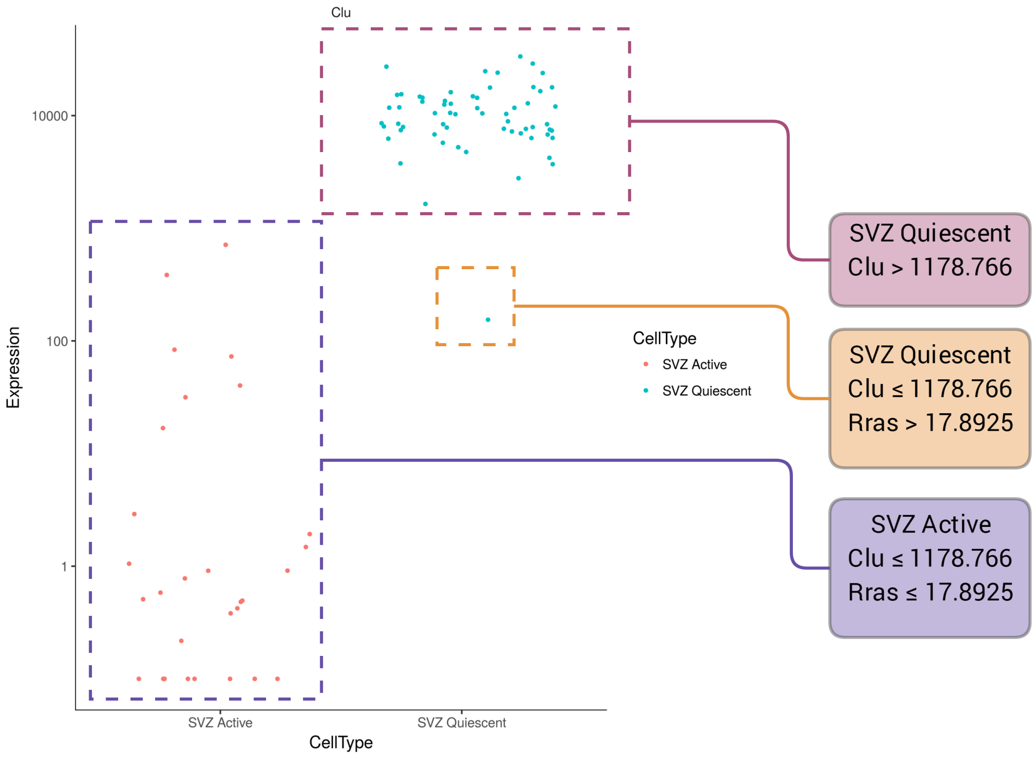


Figure 2.8: Gene Classification in the Sample Decision Tree

Chapter 3
Results/Discussion

3.1 Clustering of SVZ RNA-Seq Data Validates RAS

RAS was first tested on the SVZ data from Liorens-Bobadilla et al. [3]. The quality check analysis filtered out any genes which were not significantly expressed at a FPKM greater than 5 in more than 5 cells. The post quality check dataset contains 12103 genes. The clustering analysis of the sample cells shows the sample can be divided well into 3 groups. Further classification of the sample show that each cluster represents a different cell type: Neuroblasts (NB), Oligodendrocytes (Oligo.), and Neural Stem Cells (NSCs). This is shown in Figure 3.1. The cells were classified using the same gene markers (such as Sox9, Sox10, and Dcx) used in the Liorens-Bobadilla et al study [3]. This shows the results from the pipeline is consistent with the results shown in the original paper [3] and confirm the validity of the pipeline.

3.2 Single-Cell Analysis of NSCs in RAS reveals differences between SGZ and SVZ cells

The SVZ data was then trimmed and the NSCs were isolated. A clustering analysis shows that NSCs from the SVZ further divide into two clusters. Classification analysis using the gene markers specified in Liorens-Bobadilla et al showed that NSCs from the SVZ classify into subgroups of active and quiescent cell types as shown in Figure 3.2a. RAS was then used to process the cells in a trajectory analysis which shows the cells are ordered properly in pseudotime, showing a clear transition from the quiescent to the active cell states (Figure 3.2b). A similar processed was carried out on the SGZ data from the Shin et al. study [4]. In this study NSCs were isolated using the $Nestin - CFP^{nuc}$ genetic labeling system.

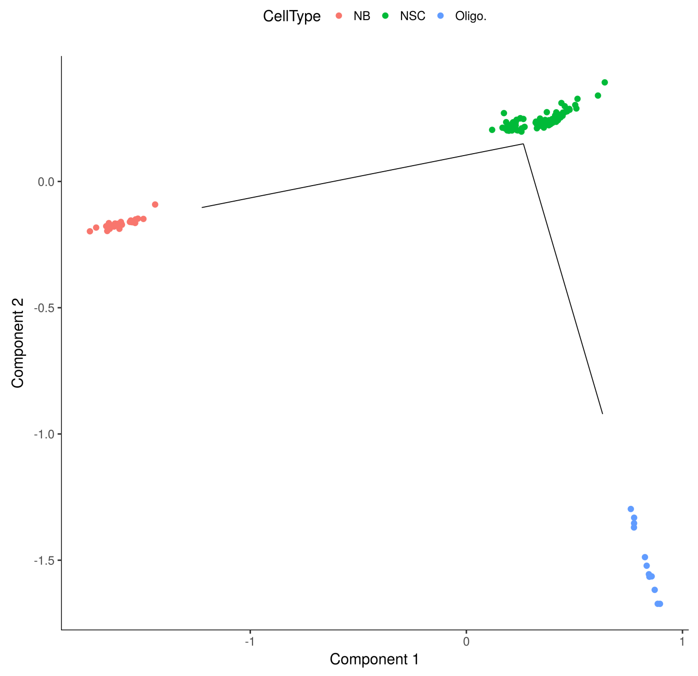
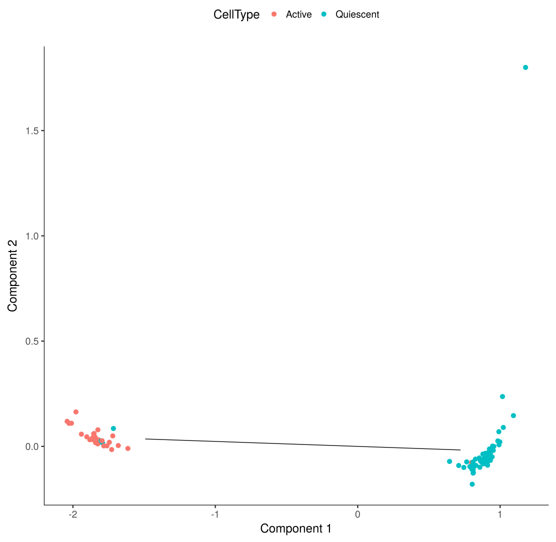
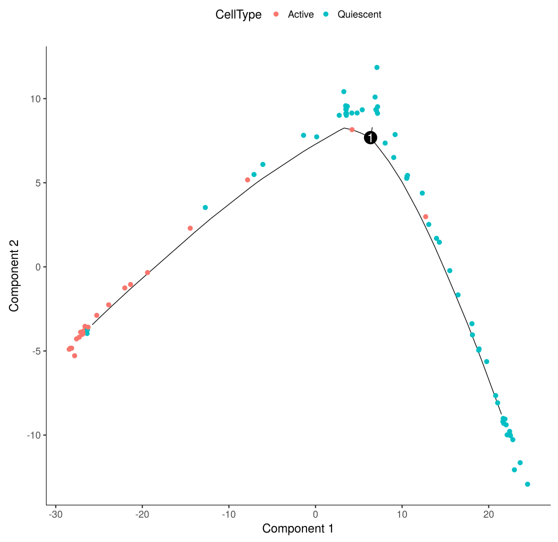


Figure 3.1: RAS identifies subpopulations of NSCs within the SVZ sample. The SVZ samples from Liorens-Bobadilla et al. were clearly classified into 3 clusters: neuroblast (NB in red), neural stem cells (NSCs in green), and oligodendrocytes (Oligo in blue), validating RAS.

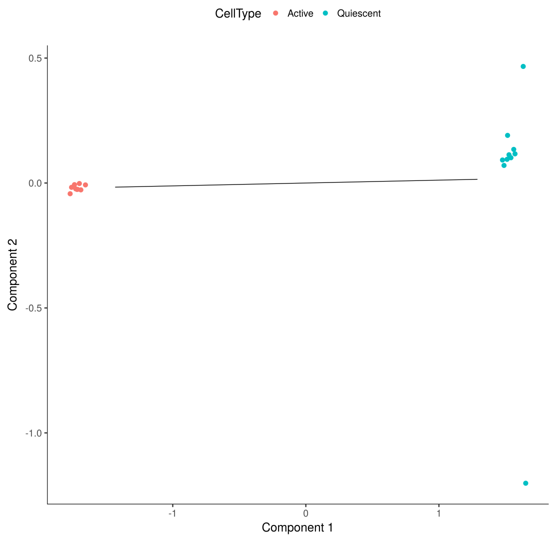
Quality check similar to that for SVZ data resulted in cells with 13088 genes. A clustering and subsequent classification analysis showed that the SGZ data also breaks up into active and quiescent cells (Figure 3.2c and Figure 3.2d). RAS was then used to combine the two studies into one experiment. Quality check of this dataset gave us cells with 11154 genes. Clustering analysis showed that the combined dataset divides into 2 clusters (Figure 3.5a). Using our classification knowledge from previous work on SGZ and SVZ cells, RAS manually classified cells into two main groups of quiescent and active based on subtypes (3.3). Each group further divides into two subgroups of SGZ and SVZ cells. This shows that a biological difference exists between SGZ and SVZ cells. A cell trajectory analysis, shows interesting pseudotime ordering between the quiescent and active cells (Figure 3.4). The active cells of both SGZ and SVZ follow the same pathway, while SGZ and SVZ are shown to follow different pathways. This indicates that active NSCs have a more similar expression pattern than quiescent cells. For further analysis, a heatmap of 150 critical genes was created using RAS. This is shown in Figure 3.7.



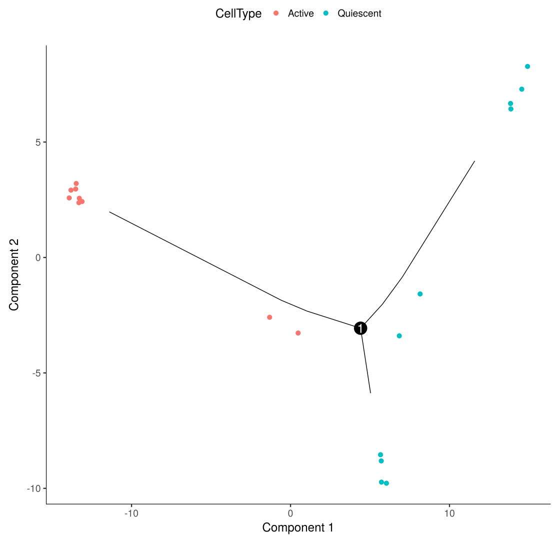
(a)



(b)



(c)



(d)

Figure 3.2: PCA analysis of clustered and pseudotime analysis of NSCs in the SGZ and SVZ. (a) NSCs in the SVZ were further clustered and classified into active and quiescent subgroups. (b) Pseudotime ordering in NSCs of the SVZ shows a clear path of transition from quiescent to activate cells. (c) NSCs of the SGZ were clustered and classified into active and quiescent subgroups. (d) Pseudotime ordering in SGZ shows 2 paths of transition form quiescent to active cells.

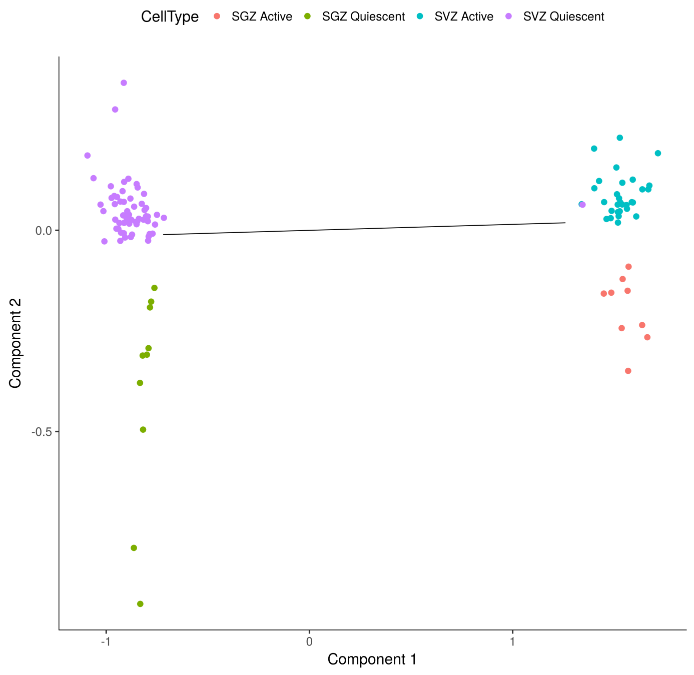


Figure 3.3: PCA analysis of combined SGZ and SVZ NSCs shows 2 main and 2 sub clusters. Quiescent and Active represent the former while SGZ and SVZ represent the latter.

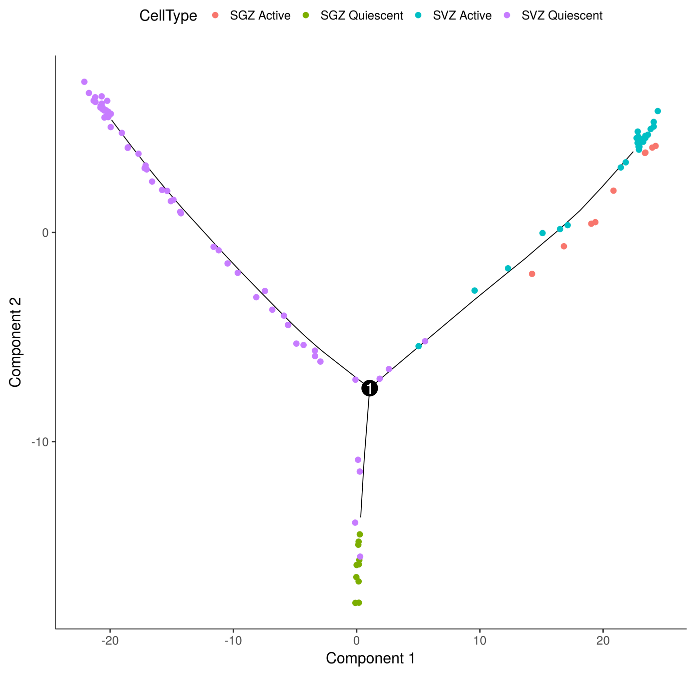


Figure 3.4: Pseudotime analysis of combined NSCs in the SGZ and SVZ shows a clear transition from quiescent to active state. Active NSCs from both SGZ and SVZ follow the same path while quiescent cells follow different paths.

3.3 Standard RNA-Seq Analysis in RAS confirms the existence of sub-clusters between SGZ and SVZ cells

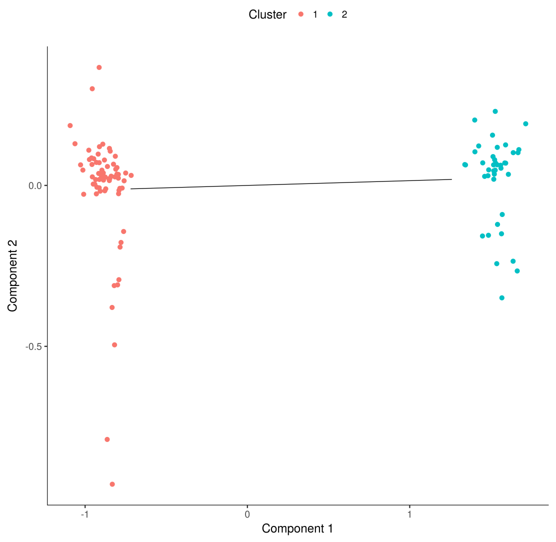
Four cells from each of the four groups of NSCs (SVZ Active, SVZ Quiescent, SGZ Active, SGZ Quiescent) were selected and analyzed through standard RNA-Seq. In Figure 3.5b, RAS shows that cells from the Liorens-Bobadilla et al. study [4] have a slightly lower quality score. A dendogram analysis shown in Figure 3.5c, confirms the previous results that active and quiescent cells cluster together, with quiescent cells being less similar across SGZ and SVZ. We then performed a decision tree analysis using RAS and found that the most significant gene for partitioning the cells into 4 groups was Apoe. The whole decision tree can be seen in Figure 3.6. We then performed differential RNA-Seq analysis on the gene Apoe. The plot in Figure 3.5d shows that SVZ quiescent cells expression of Apoe is three-fold higher than Apoe expression in the SGZ quiescent cells.

3.4 Future Work

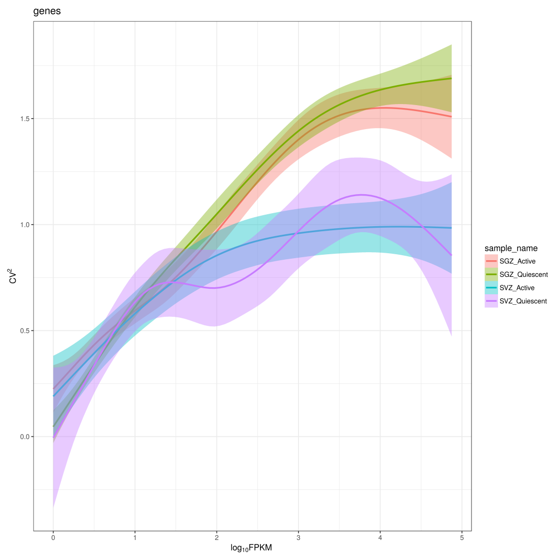
The sequencing data will be further analyzed with the pipeline to find critical genes using the combined SVZ and SGZ data. The pipeline is currently under development in a Github repository [https://github.com/nikhil/RAS] which will be made public upon completion. It contains installation instructions where the user can download and set up the program on his or her own server.

3.5 Conclusion

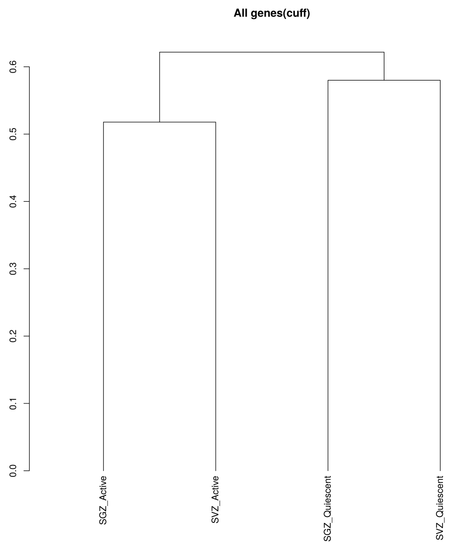
RAS has shown to be a useful tool for both single-cell and standard RNA-Seq data analysis. It was both efficient and user friendly. RAS can work with RNA-Seq data with a variable number of samples and conditions as long as the correct genome files are supplied.



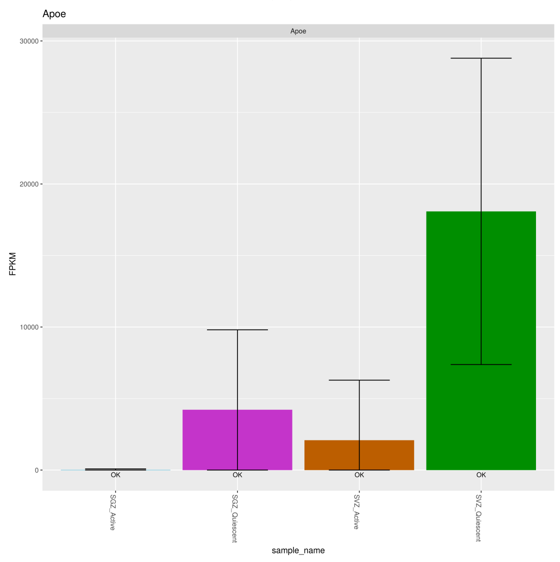
(a)



(b)



(c)



(d)

Figure 3.5: Clustering and CummeRbund analysis in RAS of the combined SGZ and SVZ NSCs. (a) PCA analysis of clustered SGZ and SVZ NSCs shows that they classify well into two groups. (b) Quality control analysis using the squared coefficient of variation. (c) Dendogram shows active and quiescent cells were classified together across SGZ and SVZ samples, with more variability in quiescent cells. (d) Bar plots of the gene Apoe gene which is expressed 3 fold more in SVZ quiescent than SGZ quiescent cells

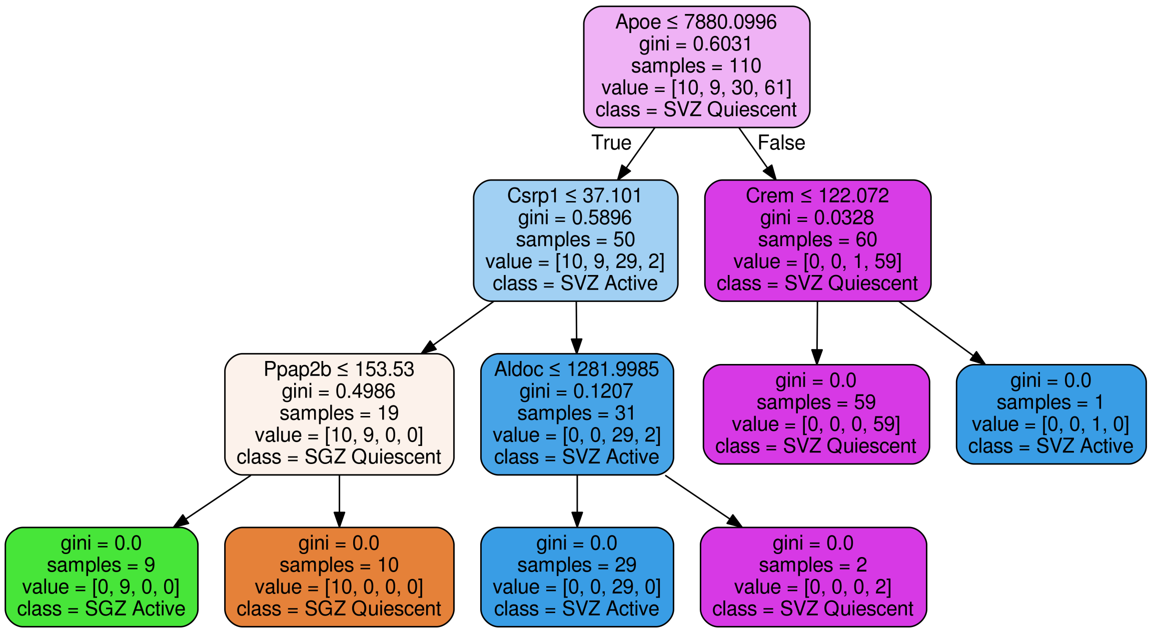


Figure 3.6: Decision tree analysis of the combined SGZ and SVZ data. Gini scores indicates how well a gene divides the samples where higher values partition better. Class indicates what cell type the gene is best at isolating. The cut-off value and gene are chosen by the decision using artificial intelligence. The value array shows the number of SGZ Quiescent, SGZ Active, SVZ Active, and SVZ Quiescent cells in the set respectively

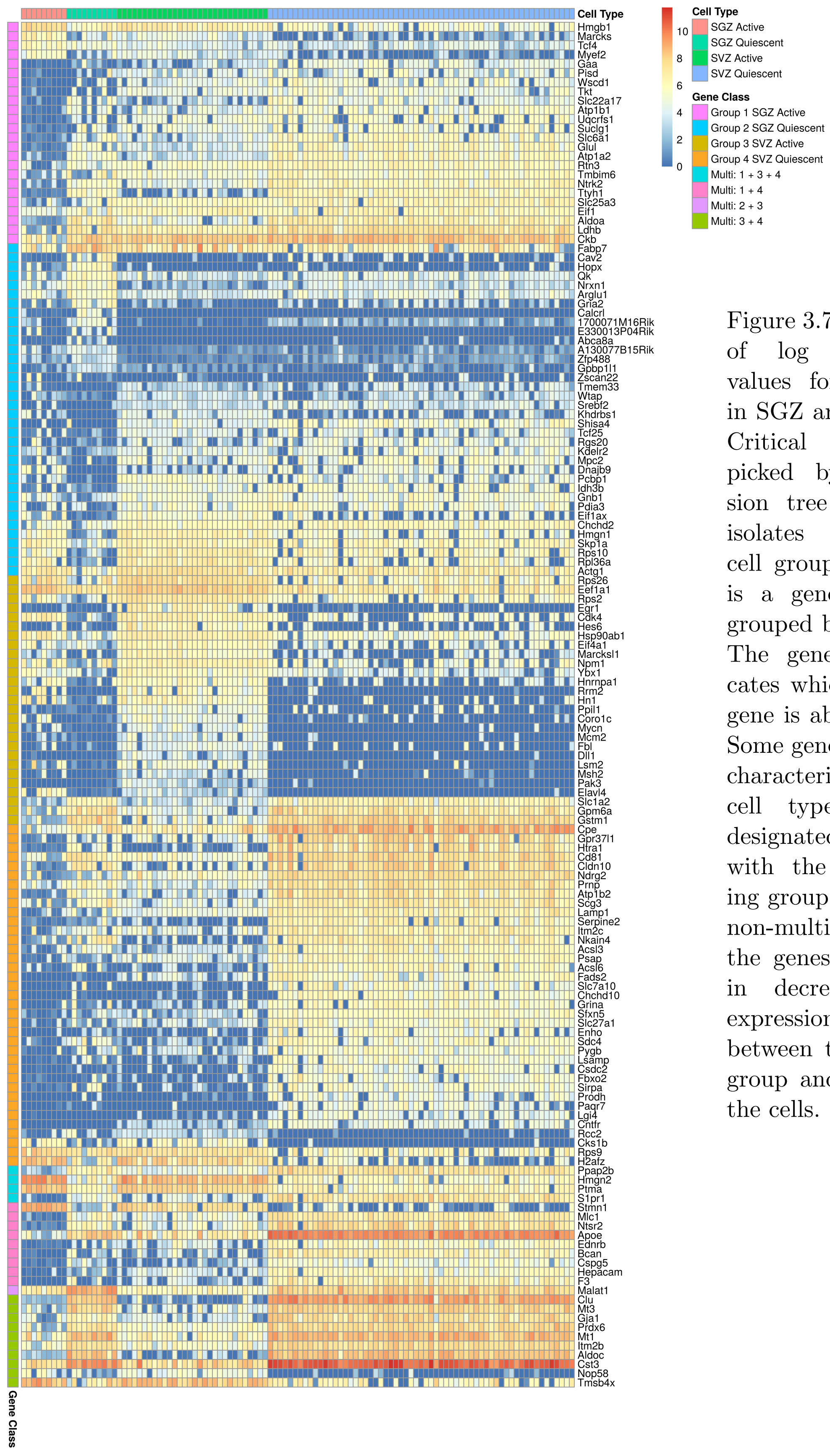
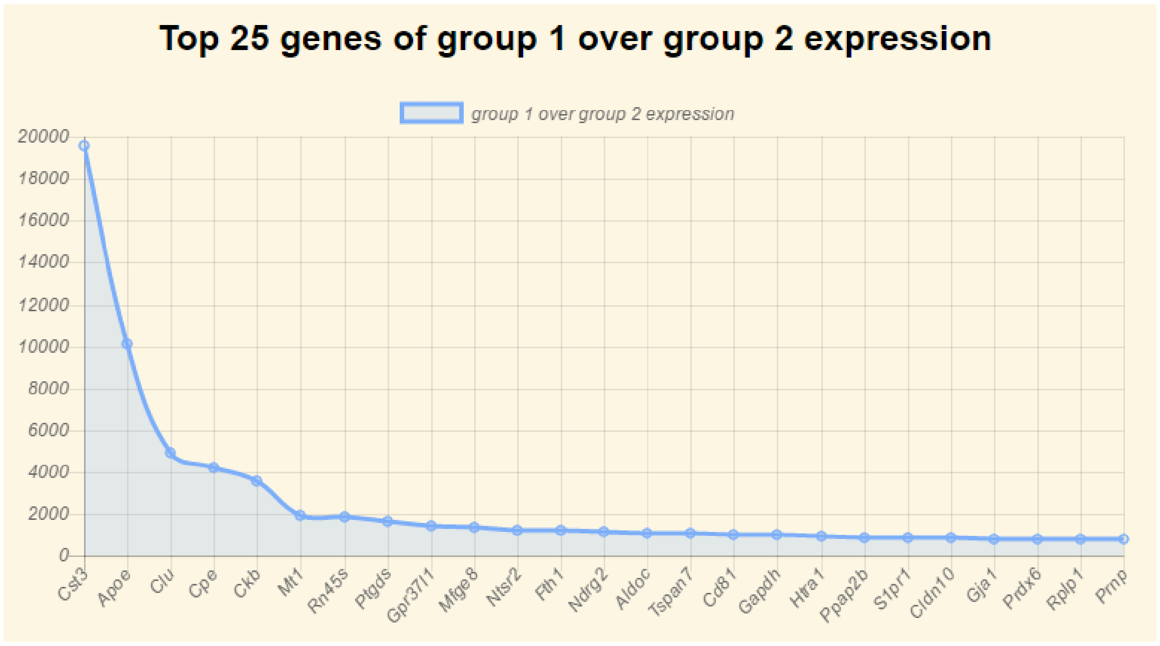
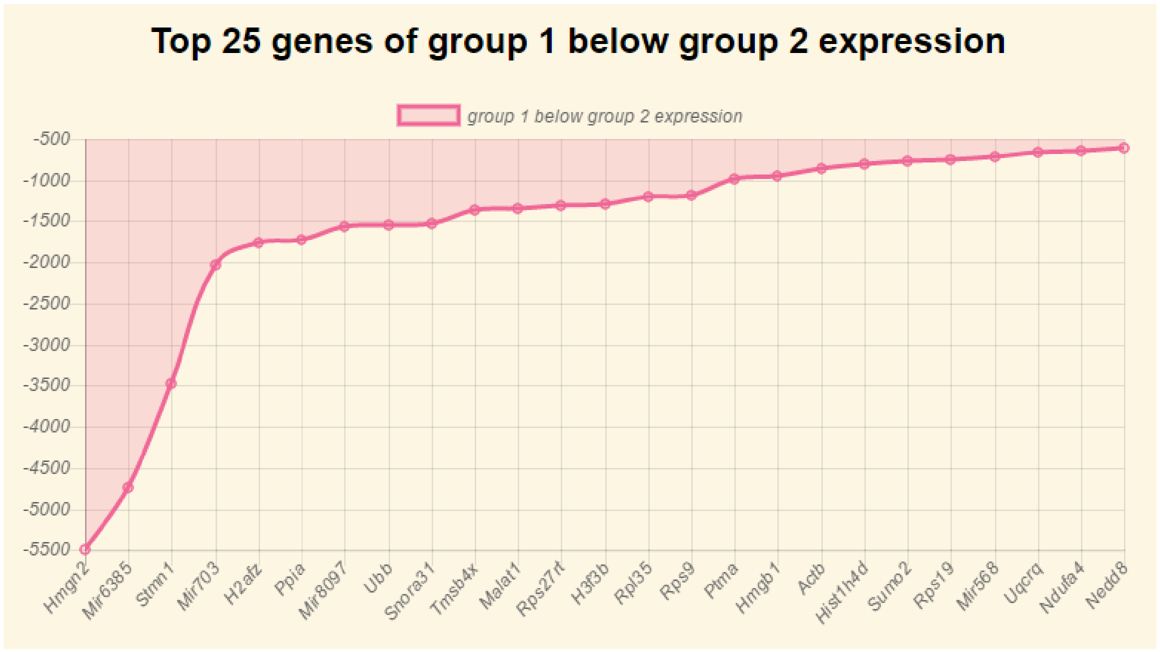


Figure 3.7: A heatmap of log transformed values for 150 genes in SGZ and SVZ cells. Critical genes were picked by the decision tree which best isolates the specific cell group. Each row is a gene which are grouped by gene class. The gene class indicates which group the gene is able to isolate. Some genes are able to characterize multiple cell types and are designated as "multi" with the corresponding group numbers. In non-multi gene classes, the genes are ordered in decreasing mean expression difference between the indicated group and the rest of the cells.

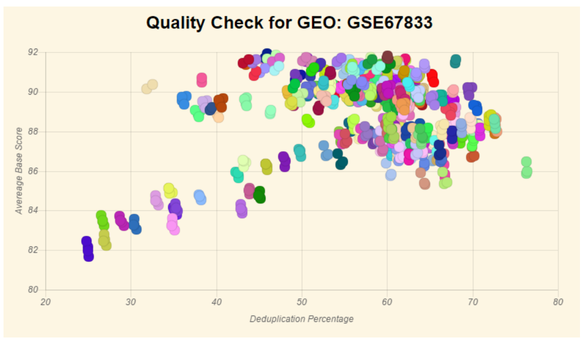


(a) Top 25 genes expressed in SVZ quiescent over SGZ quiescent

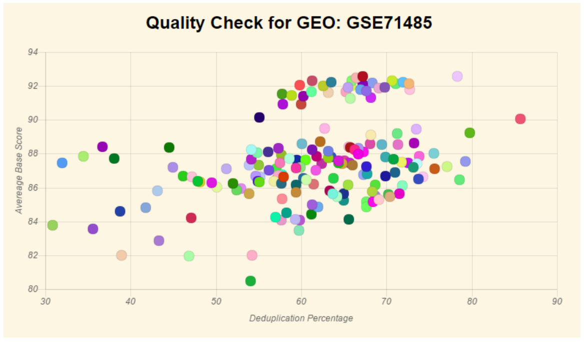


(b) Top 25 genes expressed in SGZ quiescent over SVZ quiescent

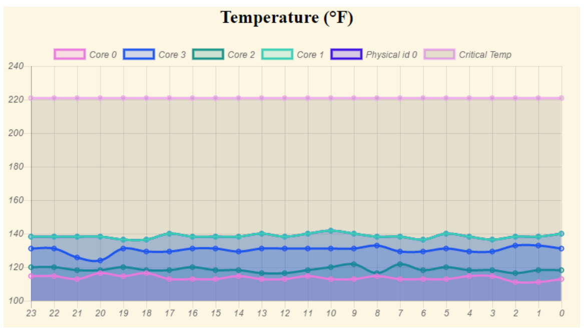
Figure 3.8: Comparison of mean gene expression values for SVZ and SVZ quiescent cells



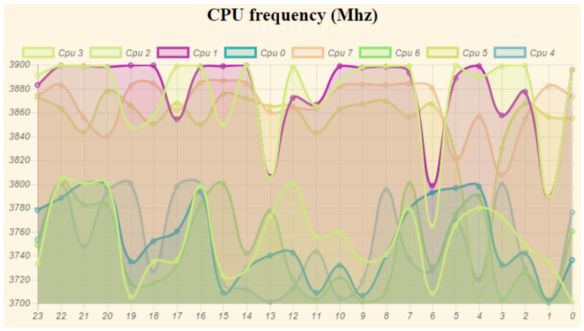
(a) Quality check scatter plot analysis of GSE67833 comparing average base score and deduplication percentage



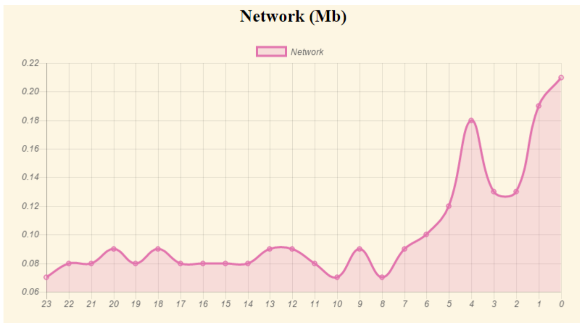
(b) Quality check scatter plot analysis of GSE71485 comparing average base score and deduplication percentage



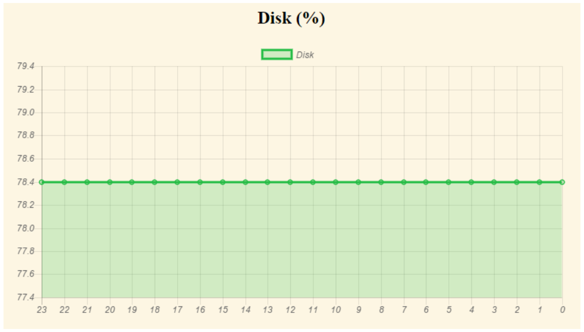
(c) Temperature readings from the last 23 minutes on all cpus with a critical temp level.



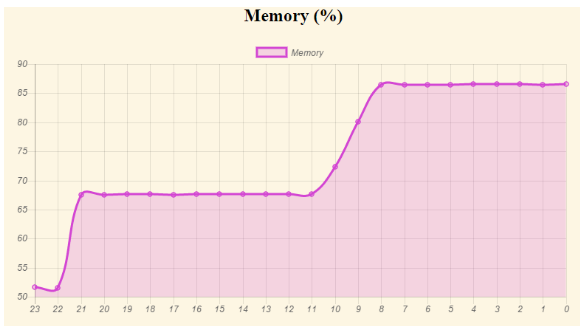
(d) CPU frequency usage of each cpu from the last 23 minutes



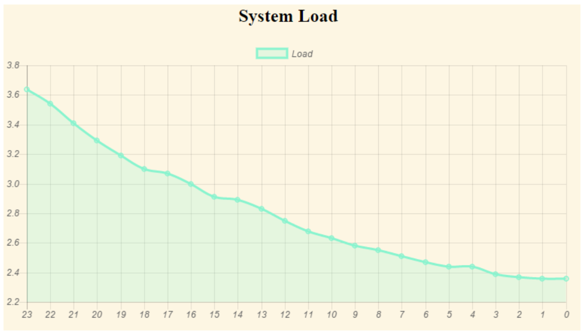
(e) Network usage from the last 23 minutes



(f) Disk usage from the last 23 minutes



(g) Memory usage from the last 23 minutes



(h) System load form the last 23 minutes

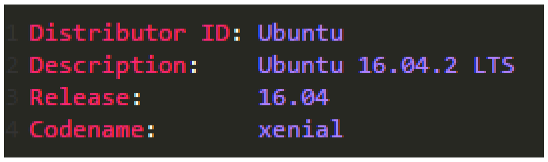
Figure 3.9: Visualization of quality control data as well as server sensor data on RAS

Appendix A
Installation

This server has been developed and shown to work on Ubuntu. The distribution used in this paper is shown in Figure A.1 User with any other distribution will need to modify parts of the installation.

A.1 Requirements

The server needs internet access to download data from the GEO. It is recommended to use a server with ample disk space and a large amount of cores. The server used in this paper has 8 cores and 2 TB of disk space.



```
Distributor ID: Ubuntu
Description:    Ubuntu 16.04.2 LTS
Release:        16.04
Codename:       xenial
```

Figure A.1: Ubuntu version used in this paper

A.2 Software packages

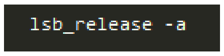
You will need to install these packages on your distro:

- python 2.7
- R
- fastq-dump
- fastQC
- TopHat

- Cufflinks
- graphviz

A.2.1 Installing MongoDB

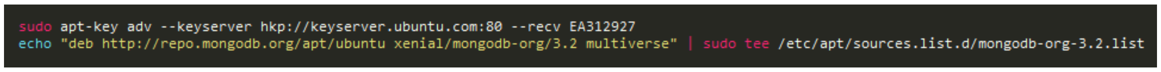
You will first need to know the code name of your Ubuntu version. You can get this with the command shown in Figure A.2. Now you need to add MongoDB to your repository.



```
lsb_release -a
```

Figure A.2: Command to view the code name of your ubuntu version

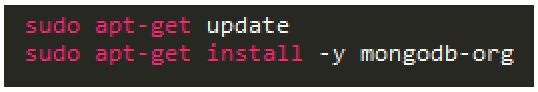
Change 'xenial' into your appropriate Ubuntu codename.



```
sudo apt-key adv --keyserver hkp://keyserver.ubuntu.com:80 --recv EA312927
echo "deb http://repo.mongodb.org/apt/ubuntu xenial/mongodb-org/3.2 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-3.2.list
```

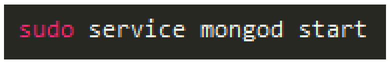
Figure A.3: Command to add MongoDB to repository

You can find more up to date instructions on:
https://docs.mongodb.com/v3.2/tutorial/install-mongodb-on-ubuntu/
Now you can install MongoDB.



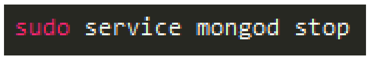
```
sudo apt-get update
sudo apt-get install -y mongodb-org
```

(a) Command to install MongoDB



```
sudo service mongod start
```

(b) Command to start MongoDB



```
sudo service mongod stop
```

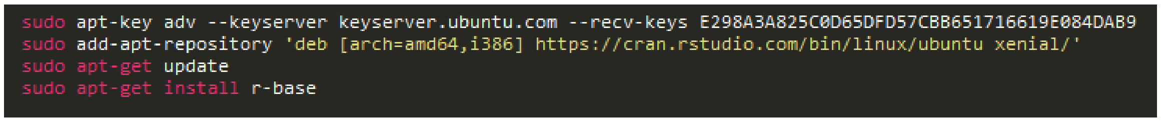
(c) Command to stop MongoDB

Figure A.4: Commands to install and start/stop MongoDB

After MongoDB is installed, you would need to start it with the command in Figure A.4b. Remember to run this command each time after the server is restarted.

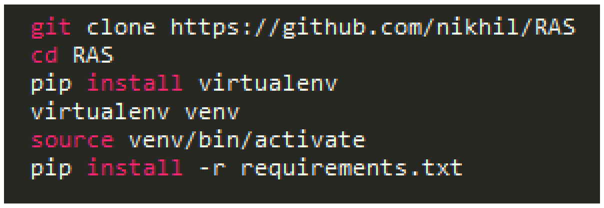
A.2.2 Installing R

You can follow the instructions from Figure A.5. Remember to replace 'xenial' with the appropriate codename. Now you can install RAS



```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD57CBB651716619E084DAB9
sudo add-apt-repository 'deb [arch=amd64,i386] https://cran.rstudio.com/bin/linux/ubuntu xenial/'
sudo apt-get update
sudo apt-get install r-base
```

Figure A.5: Command to install the latest version of R



```
git clone https://github.com/nikhil/RAS
cd RAS
pip install virtualenv
virtualenv venv
source venv/bin/activate
pip install -r requirements.txt
```

Figure A.6: Set of commands to install RAS

A.3 Python packages

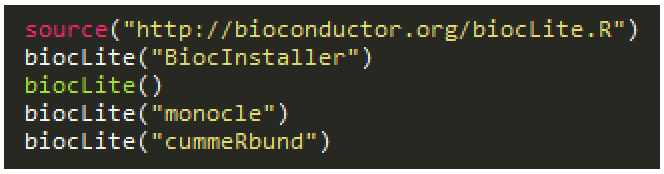
Here is the list of Python packages that you have installed with their corresponding version numbers. They are required for RAS to work.

- beautifulsoup4==4.5.1
- bx-python==0.7.3
- click==6.7
- cycler==0.10.0
- Cython==0.25.1
- enum-compat==0.0.2
- enum34==1.1.6
- filelock==2.0.7
- Flask==0.12
- greenlet==0.4.12
- itsdangerous==0.24
- Jinja2==2.9.4

- MarkupSafe==0.23
- matplotlib==1.5.3
- numpy==1.11.2
- psutil==5.1.3
- pydotplus==2.0.2
- pymongo==3.4.0
- pyparsing==2.1.10
- pysam==0.9.1.4
- python-dateutil==2.5.3
- python-engineio==1.2.3
- python-http-client==2.2.1
- pytz==2016.7
- PyYAML==3.12
- randomcolor==0.4.4.4
- requests==2.11.1
- RSeQC==2.6.4
- scikit-learn==0.18.1
- scipy==0.18.1
- sendgrid==3.6.3
- six==1.10.0
- sklearn==0.0
- Werkzeug==0.11.15

A.4 R packages

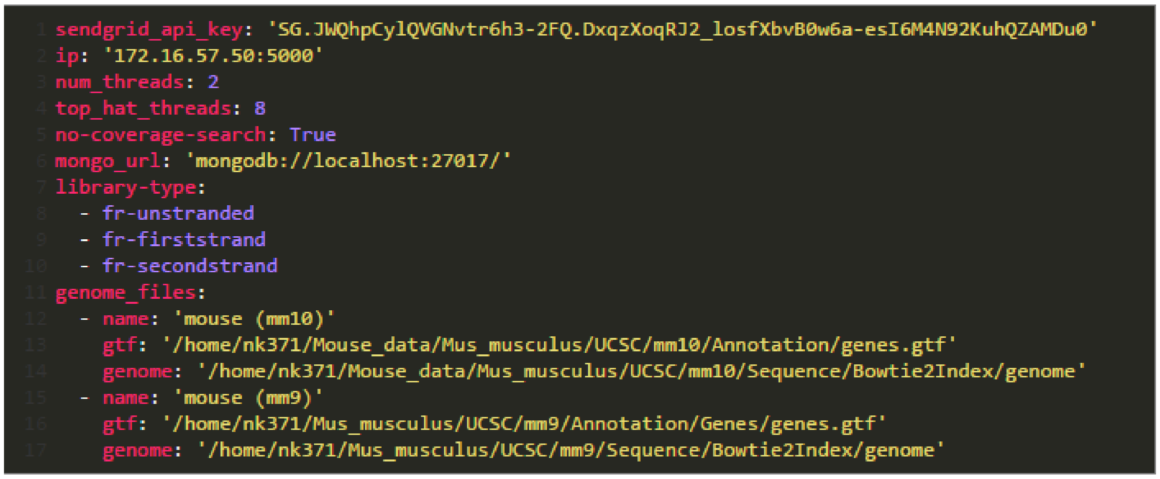
You will also need to install some R packages for RAS to work. You need to enter the R scripting environment by typing in ”R” on the terminal.



```
source("http://bioconductor.org/biocLite.R")
biocLite("BiocInstaller")
biocLite()
biocLite("monocle")
biocLite("cummeRbund")
```

Figure A.7: Commands to install the required R packages

Now you need to edit the server configuration file. The file is located in config.yaml. You will need to edit this with a text editor. A sample file is shown in Figure A.8.



```
sendgrid_api_key: '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'
ip: '172.16.57.50:5000'
num_threads: 2
top_hat_threads: 8
no-coverage-search: True
mongo_url: 'mongodb://localhost:27017/'
library-type:
  - fr-unstranded
  - fr-firststrand
  - fr-secondstrand
genome_files:
  - name: 'mouse (mm10)'
    gtf: '/home/nk371/Mouse_data/Mus_musculus/UCSC/mm10/Annotation/genes.gtf'
    genome: '/home/nk371/Mouse_data/Mus_musculus/UCSC/mm10/Sequence/Bowtie2Index/genome'
  - name: 'mouse (mm9)'
    gtf: '/home/nk371/Mus_musculus/UCSC/mm9/Annotation/Genes/genes.gtf'
    genome: '/home/nk371/Mus_musculus/UCSC/mm9/Sequence/Bowtie2Index/genome'
```

Figure A.8: YAML file to configure the RAS program

sendgrid_api_key This is the api key for a sever that is used to send emails. This is a demo key and it may be deactivated later, you should request your own at https://sendgrid.com/.

ip This is the public ip of the server. Make sure to include the port ':5000' afterwards.

num_threads This is the number of threads to process alignment jobs in parallel

top_hat_threads This is the number of threads used in each instance of Tophat

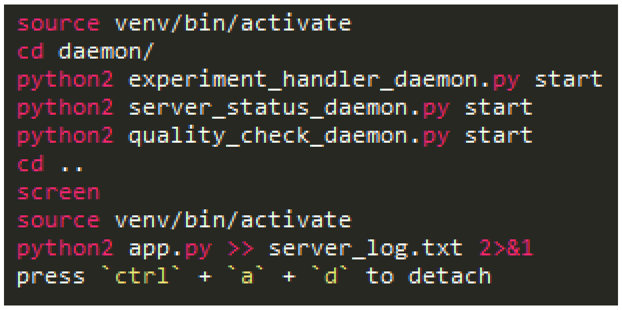
no-coverage-search A boolean to indicate if Tophat should use no-coverage-search

mongo_url This is the url where the mongodb server can be found. The default should be fine unless you are running it on a different server

library-type The options of library type the user can choose when they assign a RNA job

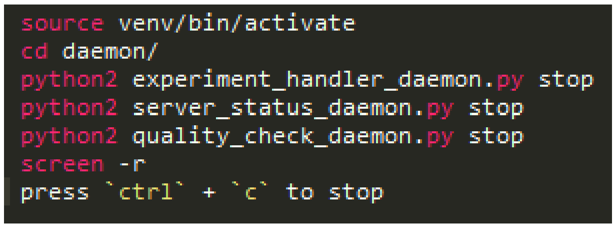
genome_files A list of dictionaries with the keys name, gtf and genome. The names define the genomes that appear on them selection bar for the user the choose. The gtf and genome paths indicate the path for the genome listed. The format should be precisely followed including spacing.

Now you need to exit R and start the daemon scripts. Make sure you are in the venv, this ensures the python packages are loaded. Now you can start the scripts by following Figure A.9a



```
source venv/bin/activate
cd daemon/
python2 experiment_handler_daemon.py start
python2 server_status_daemon.py start
python2 quality_check_daemon.py start
cd ..
screen
source venv/bin/activate
python2 app.py >> server_log.txt 2>&1
press `ctrl` + `a` + `d` to detach
```

(a) Set of commands for starting daemon scripts and server



```
source venv/bin/activate
cd daemon/
python2 experiment_handler_daemon.py stop
python2 server_status_daemon.py stop
python2 quality_check_daemon.py stop
screen -r
press `ctrl` + `c` to stop
```

(b) Set of commands for stopping daemon scripts and server

Figure A.9: Commands to start/stop daemon scripts and server

Remember to re-start the scripts after the server is restarted.

Appendix B

Instructions

You can access the pipeline by opening your browser and typing in [your sever ip]:5000 where your server ip is the ip you specified in the configuration file. In this page you can choose between going to the standard, single-cell, or GEO quality check workflows, viewing the status of the server or job queues.

B.1 Single-Cell RNA-Seq

You can see the workflow page on Figure C.1c

B.1.1 Align Data

The workflow lists the available functions in order. They must be followed in that order. You can't run a classification function such as differential gene analysis without first classification the data. the first step is to align and prepare the data. Here you can either assign a job or combine experiments which have already been aligned on RAS. In the assign job page, you will notice that it has two dynamic forms. You can add or remove conditions and you can add or remove samples. It is worth noting the difference between 'condition' and 'sample'. These two terms get used interchangeably between software which could be a source of confusion. In RAS, 'sample' stands for 1 sequence or 1 replicate. In other words one sequence is once file. 'Condition' stands for a collection of samples which indicate a certain state such as quiescent. For each sample you can upload a '.sra' file from your local computer. You can also paste links for a url upload. The url upload must be from the GEO and include the 'ftp://' protocol in the link. You need to link the file not the folder.

B.1.2 Analyze significant genes

For getting significant genes, you need to create two groups by indicating the cell number to be included in each group. A range of numbers can be entered in a certain format. You can use '-' (e.g. 2-10) for an inclusive range and commas to add numbers. For example 1,2,4-7 will include [1,2,4,5,6,7] for one group. A samples table is provided for your reference.

B.1.3 Quality Control

You will notice that in many of the function you need to enter the experiment id before seeing the input form. This is needed to appropriately show the samples table. For the input form, you need to enter the range of mRNA to use. All cells outside the range will be removed. To pass all cells use: 'inf' for the upper bound, '0' for the lower bound and '0' for minimum expression and '0' for minimum expression. Quality control also works by eliminating genes that are not significantly expressed in a certain number of cells. The gene expression and number of cells correspond to the significance and number accordingly. For example, a minimum gene expression value of 2 and minimum cells value of 4 will remove all genes that are not expressed at a FPKM value over 2 in over 4 cells. A samples table is provided for your reference.

B.1.4 Post-quality control Analysis

After quality control you can Trim, cluster the data or process it through a heatmap or decision tree. For trim, the range format can be used. All of the cells outside the range will be excluded. After trim, the only way to get the data back is to run through quality control again. For a decision tree, you have a dynamic form where you can enter each groups range and corresponding label. The manual heatmap is called "manual" because it allows the users to manually specify groups for cells. The number of genes indicates how many significant genes to use. The gini value is a measure of information gain for each gene that divides the dataset at a determined point. The value depends on the dynamics of the dataset. A sample with a large number of one group can bias the dataset to choose more genes that isolate that group. A lower gini value allows the program to look genes for other groups at the expense of gene significance. Make sure the gene number is in a feasible range. For clustering analysis, simply enter the number of samples to analyze. Unfortunatly, monocle uses an iterative program. THis leads to cases where different runs can generate different results due to different steady states. This could be fixed by running better quality control.

B.1.5 Classification

You can classify the cells by gene expression or by manual selection. In the way of gene expression, the form allows you to specify conditions for each group. In this context, the conditions are logical statements which can be combined by an AND or an OR. The label name is what the condition will be called. The manual selection method uses the similar dynamic form where groups can be created using the label and range format. After each classification analysis method, the function shows a pca plot of the cells where each condition is represented by a different color.

B.1.6 Post-classification analysis

Get classified heatmeap is similar to get manual heatmap, the only difference is that the groups are automatically picked out based on the classification of the dataset. Get gene pseudotime performs a pseudotime analysis of the genes and cells. The user needs to specify the number of dimensions and a list of genes. Differnetial gene analysis simply shows the gene value of each cell where the class of the cell is indicated. Multiple genes can be listed. Cell trajectory orders the cells in pseudotime. You will need to specify the dimensions for this step as well.

B.2 Standard RNA-Seq

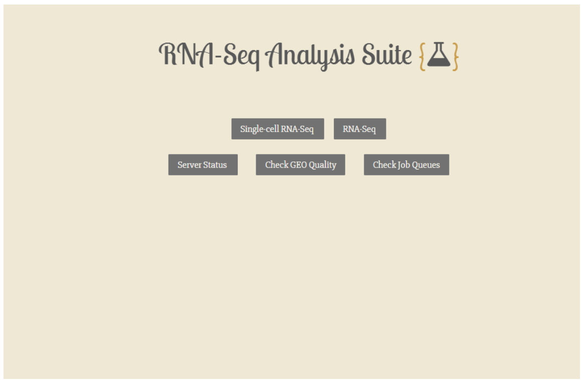
The standard RNA-Seq work flow is roughly similar to the single-cell workflow. The first step will assign a standard RNA-Seq workflow. The second step will create the cuffset Datanase for further analysis. The third step will show the quality control plots of the cells. This will not alter the dataset, it just shows the quality. In post quality control analysis, the user can get a dendogram or get a comparison of a certain gene expression. The fifth step, gets significant genes either by expression or differential comparison. For differential analysis you need to indicate the names of 2 samples, while by expression will look at all the samples. Both methods need an alpha value. This controls the significance of of the genes. Alpha must be less than 1 and greater than 0. If you set alpha too low, you will get an error because you will not get any genes. After a significant gene dataset is generated the number of genes chosen is returned. You can do further analysis with the significant genes such as generate a heatmap, get similar genes, and get a k means gene cluster. It is recommended to have a small gene subset for the heatmap labels to legible.

B.3 Check GEO Quality

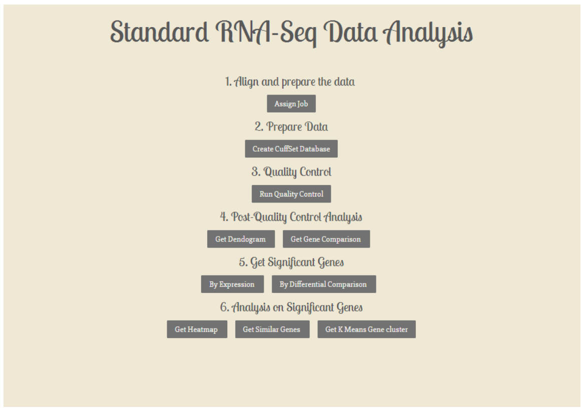
In this function you can run or analyze the results of previous runs. It is recommended not to have too many of these jobs. Each job downloads all the data from the GEO server which is often large in magnitude. This can overload the server and make it possible for GEO to block your server's access to their server. The function waits 1.5 seconds after each download. When entering the GEO number remember to include the letters before it as well.

Appendix C

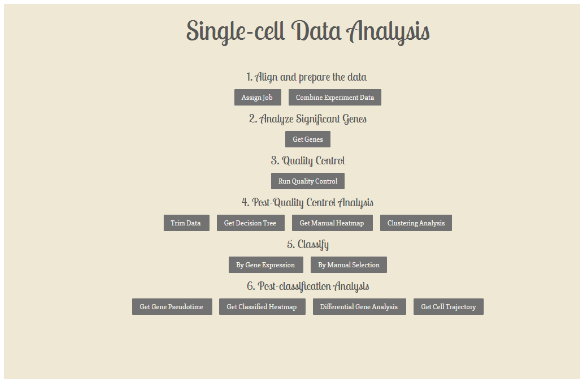
Screenshots of RAS



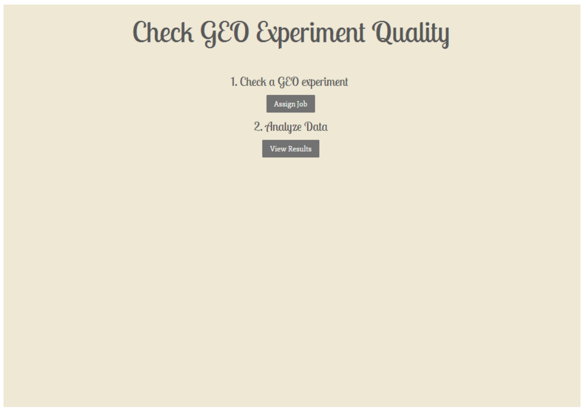
(a)



(b)

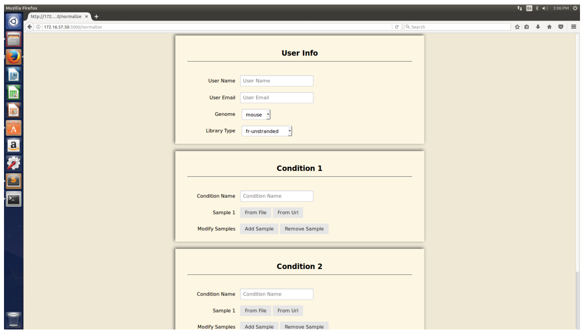


(c)



(d)

Figure C.1: RAS main webpages. (a) Main webpage with buttons to workflows and monitoring pages (b) Standard RNA-Seq workflow webpage (c) Single-Cell workflow webpage (d) Webpage for quality check analysis of GEO experiments.



User Info

User Name	User Name
User Email	User Email
Genome	mouse
Library Type	fr-unstranded

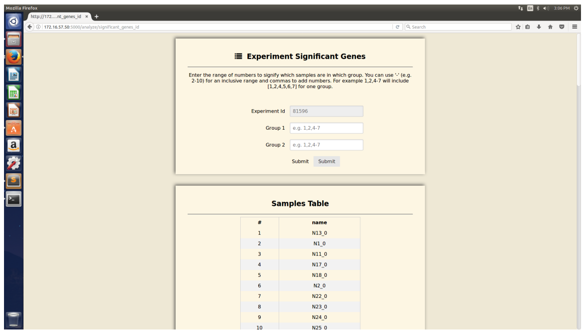
Condition 1

Condition Name	Condition Name
Sample 1	From File / From Url
Modify Samples	Add Sample / Remove Sample

Condition 2

Condition Name	Condition Name
Sample 1	From File / From Url
Modify Samples	Add Sample / Remove Sample

(a)


Mozilla Firefox
http://172....nt_genes_id
172.16.57.50:5000/analyze/significant_genes_id
Search
Submit

Experiment Significant Genes

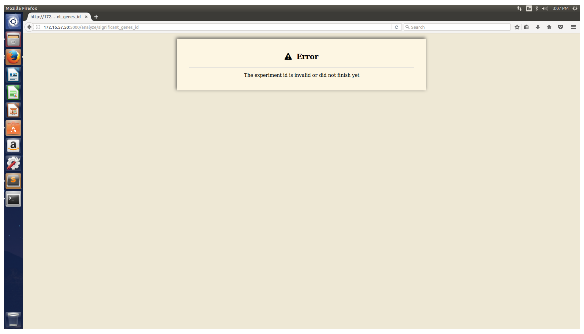
Enter the range of numbers to signify which samples are in which group. You can use '-' (e.g. 2-10) for an inclusive range and commas to add numbers. For example 1,2,4-7 will include [1,2,4,5,6,7] for one group.

Experiment Id	81596
Group 1	e.g. 1,2,4-7
Group 2	e.g. 1,2,4-7
Submit	Submit

Samples Table

#	name
1	N13_0
2	N1_0
3	N11_0
4	N17_0
5	N18_0
6	N2_0
7	N22_0
8	N23_0
9	N24_0
10	N25_0

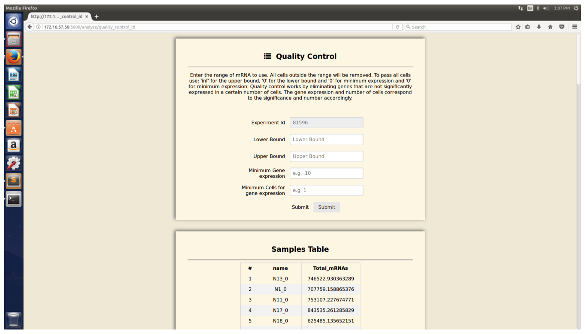
(b)


Mozilla Firefox
http://172....nt_genes_id
172.16.57.50:5000/analyze/significant_genes_id
Search

Error

The experiment id is invalid or did not finish yet

(c)


Mozilla Firefox
http://172.1...._control_id
172.16.57.50:5000/analyze/quality_control_id
Search
Submit

Quality Control

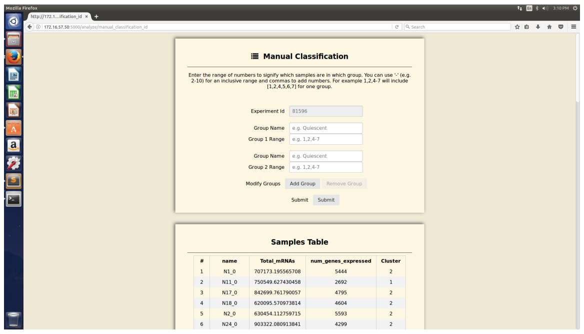
Enter the range of mRNA to use. All cells outside the range will be removed. To pass all cells use: 'inf' for the upper bound, '0' for the lower bound and '0' for minimum expression and '0' for minimum expression. Quality control works by eliminating genes that are not significantly expressed in a certain number of cells. The gene expression and number of cells correspond to the significance and number accordingly.

Experiment Id	81596
Lower Bound	Lower Bound
Upper Bound	Upper Bound
Minimum Gene expression	e.g. .10
Minimum Cells for gene expression	e.g. 1
Submit	Submit

Samples Table

#	name	Total_mRNAs
1	N13_0	746522.930363289
2	N1_0	707759.158865376
3	N11_0	753107.227674771
4	N17_0	843535.261285829
5	N18_0	625485.135652151

(d)


Mozilla Firefox
http://172.1....ification_id
172.16.57.50:5000/analyze/manual_classification_id
Search
Add Group
Remove Group
Submit

Manual Classification

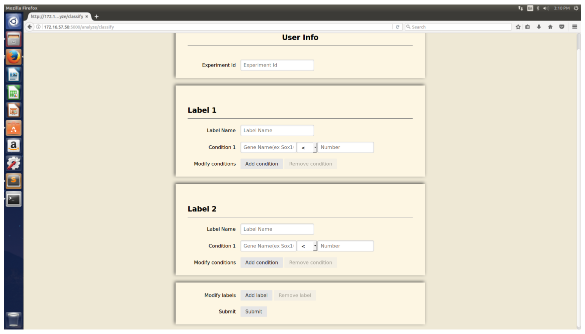
Enter the range of numbers to signify which samples are in which group. You can use '-' (e.g. 2-10) for an inclusive range and commas to add numbers. For example 1,2,4-7 will include [1,2,4,5,6,7] for one group.

Experiment Id	81596
Group Name	e.g. Quiescent
Group 1 Range	e.g. 1,2,4-7
Group Name	e.g. Quiescent
Group 2 Range	e.g. 1,2,4-7
Modify Groups	Add Group / Remove Group
Submit	Submit

Samples Table

#	name	Total_mRNAs	num_genes_expressed	Cluster
1	N1_0	707173.195565708	5444	2
2	N11_0	750549.627430458	2692	1
3	N17_0	842699.761790057	4795	2
4	N18_0	620095.570973814	4604	2
5	N2_0	630454.112759715	5593	2
6	N24_0	903322.080913841	4299	2

(e)



User Info

Experiment Id	Experiment Id

Label 1

Label Name	Label Name
Condition 1	Gene Name(ex Sox1 < Number
Modify conditions	Add condition / Remove condition

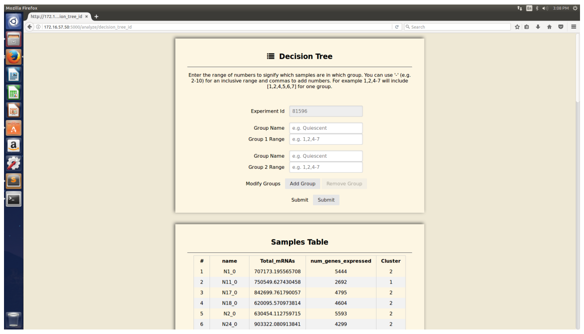
Label 2

Label Name	Label Name
Condition 1	Gene Name(ex Sox1 < Number
Modify conditions	Add condition / Remove condition

Modify labels	Add label / Remove label
Submit	Submit

(f)

Figure C.2: First set of RAS assorted screenshots.(a) Dynamic form used to input data for either single-cell or standard RNA-Seq analysis (b) Input form for significant genes between any two groups of cells (c) A sample error page when the user puts the wrong input (d) Input form for quality control of single-cell data (e) Input form for manual classification with a dynamic form from group labels and names (f) Dynamic form for classification based on gene conditions.



Decision Tree

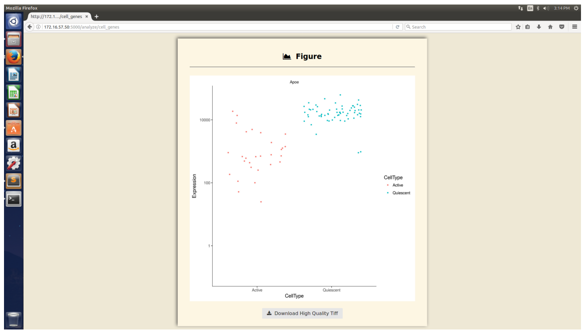
Enter the range of numbers to signify which samples are in which group. You can use '-' (e.g. 2-10) for an inclusive range and commas to add numbers. For example 1,2,4-7 will include [1,2,4,5,6,7] for one group.

Experiment Id	81596
Group Name	e.g. Quiescent
Group 1 Range	e.g. 1,2,4-7
Group Name	e.g. Quiescent
Group 2 Range	e.g. 1,2,4-7
Modify Groups	Add Group · Remove Group
Submit	Submit

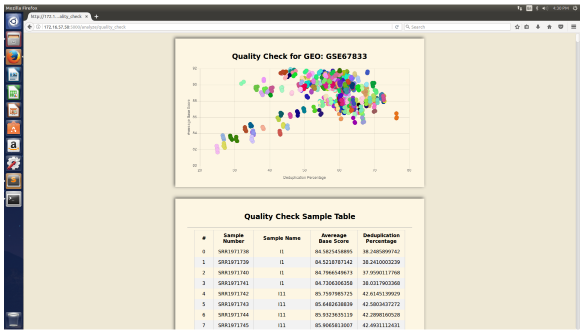
Samples Table

#	name	Total_mRNAs	num_genes_expressed	Cluster
1	N1_0	707173.195565708	5444	2
2	N11_0	750549.627430458	2692	1
3	N17_0	842699.761790057	4795	2
4	N18_0	620095.570973814	4604	2
5	N2_0	630454.112759715	5593	2
6	N24_0	903322.080913841	4299	2

(a)



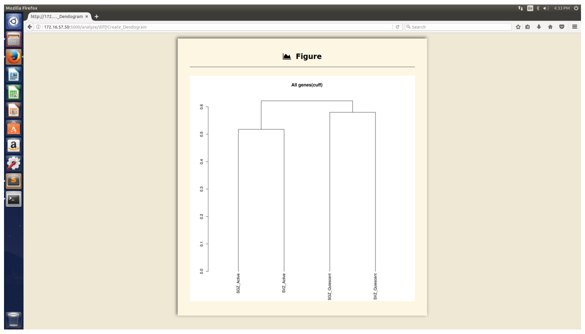
(b)



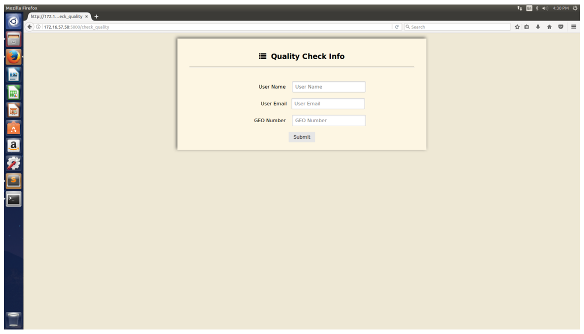
Quality Check Sample Table

#	Sample Number	Sample Name	Avereage Base Score	Deduplication Percentage
0	SRR1971738	I1	84.5825458895	38.2485899742
1	SRR1971739	I1	84.5218787142	38.2410003239
2	SRR1971740	I1	84.7966549673	37.9590117768
3	SRR1971741	I1	84.7306306358	38.0317903368
4	SRR1971742	I11	85.7597985725	42.6145139929
5	SRR1971743	I11	85.6482638839	42.5803437272
6	SRR1971744	I11	85.9323635119	42.2898160528
7	SRR1971745	I11	85.9065813007	42.4931112431

(c)



(d)

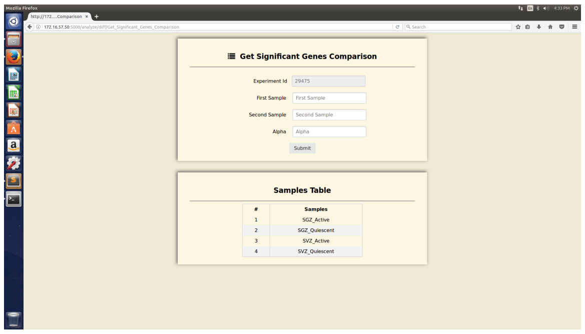


Quality Check Info

User Name	User Name
User Email	User Email
GEO Number	GEO Number

Submit

(e)



Get Significant Genes Comparison

Experiment Id	29475
First Sample	First Sample
Second Sample	Second Sample
Alpha	Alpha

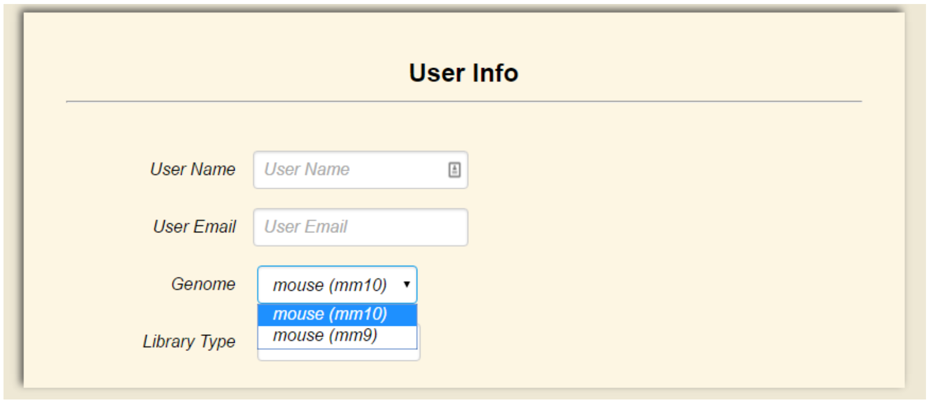
Submit

Samples Table

#	Samples
1	SGZ_Active
2	SGZ_Quiescent
3	SVZ_Active
4	SVZ_Quiescent

(f)

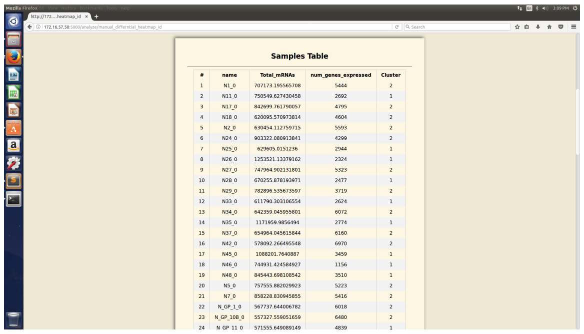
Figure C.3: Second set of RAS assorted screenshots.(a) Dynamic input form for creating a decision tree between a set of groups (b) A sample response page with a .png figure and a .tiff image download link (c) Scatter plot analysis of GEO quality check data (d) A figure response page from the standard RNA-Seq workflow (e) Input form for a quality check job on a GEO number (f) Input form for retrieving significant genes from a standard RNA-Seq dataset



User Info

User Name	User Name
User Email	User Email
Genome	mouse (mm10)
Library Type	

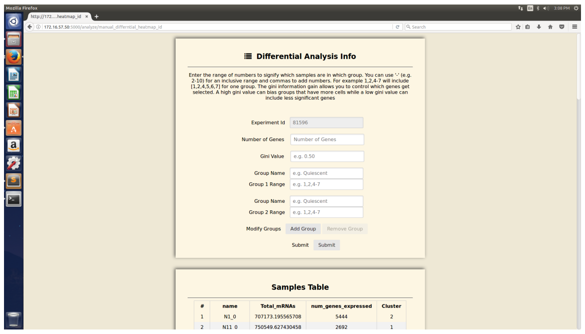
(a)



Samples Table

#	name	Total_mRNAs	num_genes_expressed	Cluster
1	N1_0	707173.195565708	5444	2
2	N11_0	750549.627430458	2692	1
3	N17_0	842699.761790057	4795	2
4	N18_0	620095.570973814	4604	2
5	N2_0	630454.112759715	5593	2
6	N24_0	903322.080913841	4299	2
7	N25_0	629605.0151236	2944	1
8	N26_0	1253521.13379162	2324	1
9	N27_0	747964.902131801	5323	2
10	N28_0	670255.878193971	2477	1
11	N29_0	782896.535673597	3719	2
12	N33_0	611790.303106554	2624	1
13	N34_0	642359.045955801	6072	2
14	N35_0	1171959.9856494	2774	1
15	N37_0	654964.045615844	6160	2
16	N42_0	578092.266495548	6970	2
17	N45_0	1088201.7640887	3459	1
18	N46_0	744931.424584927	1156	1
19	N48_0	845443.698108542	3510	1
20	N5_0	757555.882029923	5223	2
21	N7_0	858228.830945855	5416	2
22	N_GP_1_0	567737.644006782	6018	2
23	N_GP_10B_0	557327.559051659	6480	2
24	N_GP_11_0	571555.649089149	4839	1

(b)



Differential Analysis Info

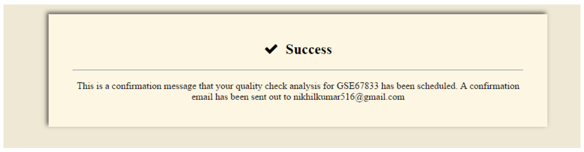
Enter the range of numbers to signify which samples are in which group. You can use '-' (e.g. 2-10) for an inclusive range and commas to add numbers. For example 1,2,4-7 will include [1,2,4,5,6,7] for one group. The gini information gain allows you to control which genes get selected. A high gini value can bias groups that have more cells while a low gini value can include less significant genes

Experiment Id	81596
Number of Genes	Number of Genes
Gini Value	e.g. 0.50
Group Name	e.g. Quiescent
Group 1 Range	e.g. 1,2,4-7
Group Name	e.g. Quiescent
Group 2 Range	e.g. 1,2,4-7

Samples Table

#	name	Total_mRNAs	num_genes_expressed	Cluster
1	N1_0	707173.195565708	5444	2
2	N11_0	750549.627430458	2692	1

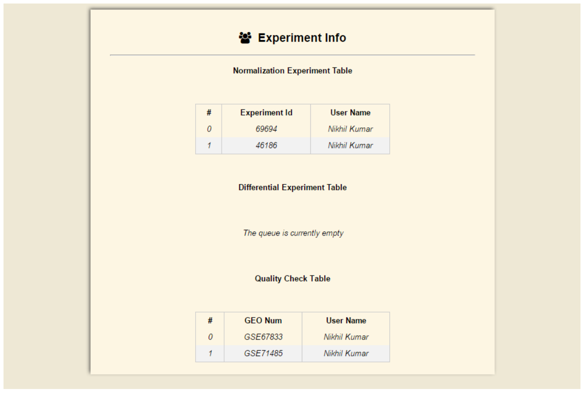
(c)



Success

This is a confirmation message that your quality check analysis for GSE67833 has been scheduled. A confirmation email has been sent out to nikhilkumar516@gmail.com

(d)



Experiment Info

Normalization Experiment Table

#	Experiment Id	User Name
0	69694	Nikhil Kumar
1	46186	Nikhil Kumar

Differential Experiment Table

The queue is currently empty

Quality Check Table

#	GEO Num	User Name
0	GSE67833	Nikhil Kumar
1	GSE71485	Nikhil Kumar

(e)

Figure C.4: Third set of RAS assorted screenshots.(a) Custom genomes added to RNA-Seq job input form (b) The samples table which appears is many of the input forms (c) Dynamic form for group labels and ranges used to generate a differential heatmap.(d) A sample success response (e) Webpage showing the status of Normilization (single-cell), differential (standard), and quality check job queues

Appendix D

Code Snippets

The project is over 2 thousand lines of code. If you would like to see the scripts in more detail you can do so at: https://github.com/nikhil/RAS. In this chapter, I will highlight select sections of the program.

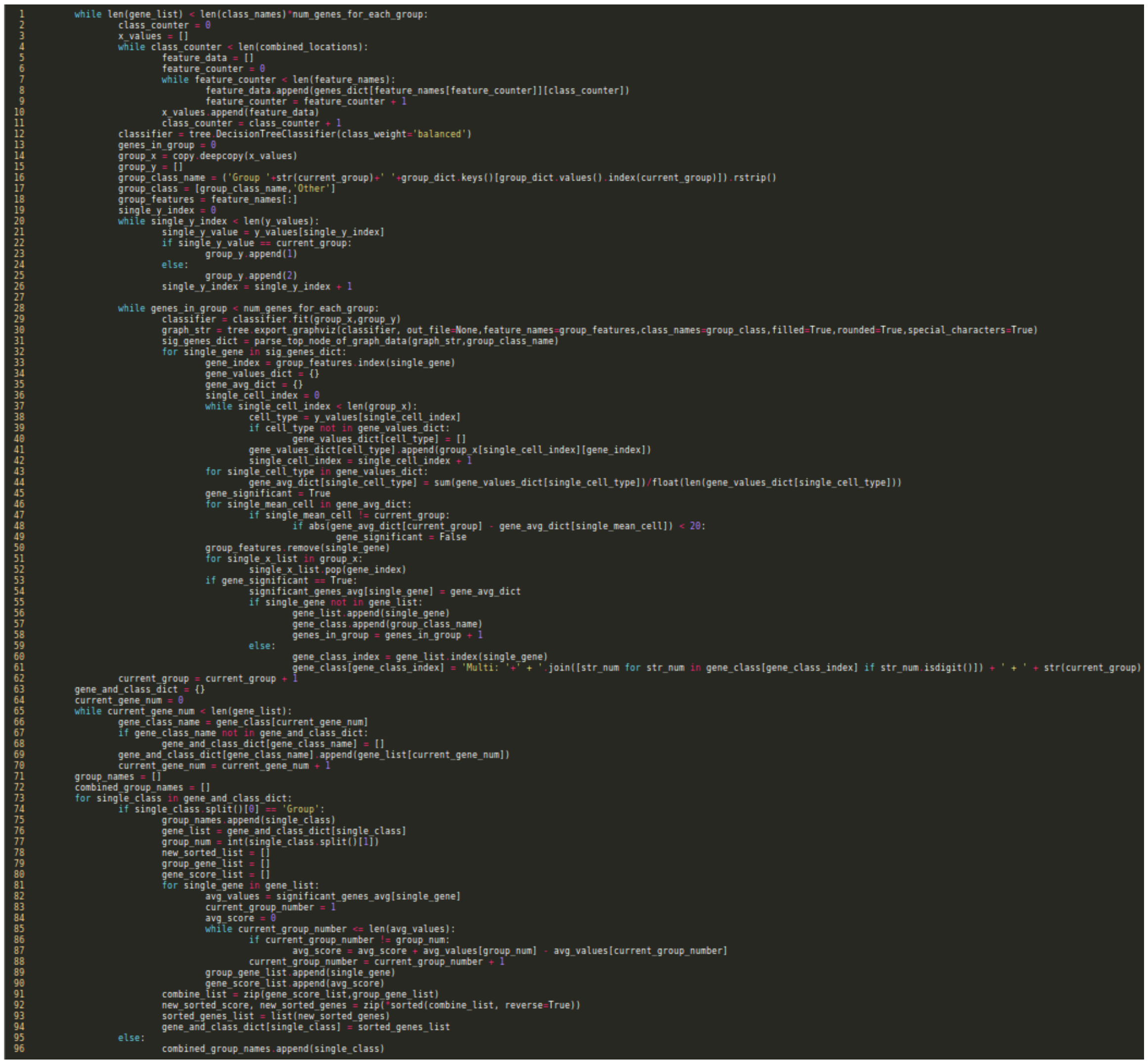
D.1 Heatmap function

D.1.1 Python

The heatmap function would take in the experiment id and the number of genes by the user. It would then process the cell ranges and their corresponding labels. The program returns an error if there was a mistake found in the format of the ranges. The post-quality control expression and sample table are loaded. A matrix of x values is created where the rows are each cell and the columns are the expression of each gene. The y values are the corresponding group labels of all the cells. For each cell class, the current class and the other classes are separated into two groups. The function is then run through a loop of n times where n is the number of significant genes specified by the user divided by the number of groups. At each iteration:

1. The decision tree is trained
2. The custom method parse_graph_data is called
3. The tree data is parsed using string manipulation
4. The gene is the top node is returned
5. The gene is stored and removed from the training set.

The appropriate R script is called. The figure is moved into the static folder to be accessible by the user. A figure link is generated and returned to the user.



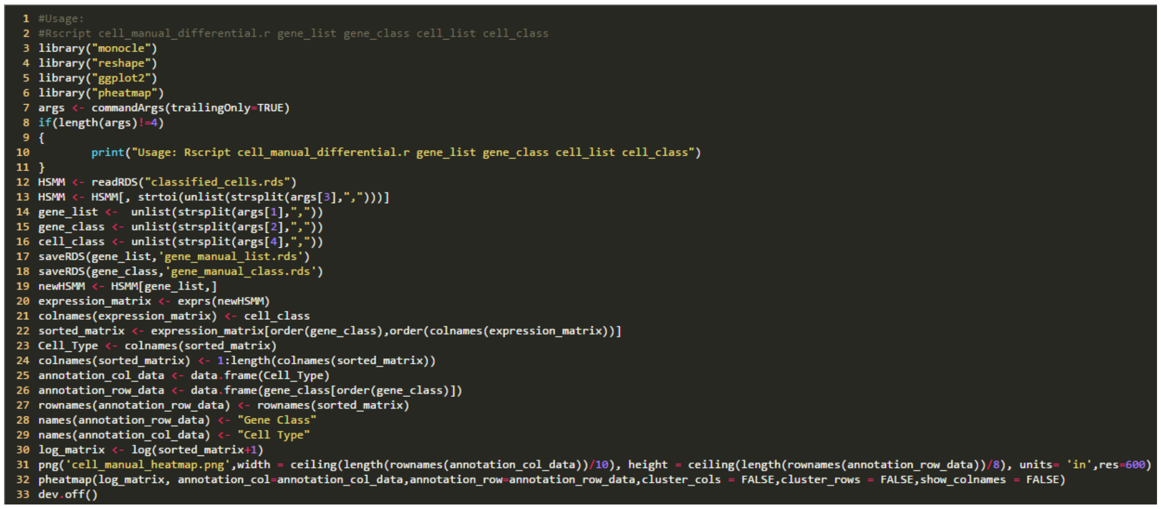
```
    while len(gene_list) < len(class_names)*num_genes_for_each_group:
        class_counter = 0
        x_values = []
        while class_counter < len(combined_locations):
            feature_data = []
            feature_counter = 0
            while feature_counter < len(feature_names):
                feature_data.append(genes_dict[feature_names[feature_counter]][class_counter])
                feature_counter = feature_counter + 1
            x_values.append(feature_data)
            class_counter = class_counter + 1
        classifier = tree.DecisionTreeClassifier(class_weight='balanced')
        genes_in_group = 0
        group_x = copy.deepcopy(x_values)
        group_y = []
        group_class_name = ('Group '+str(current_group)+' '+group_dict.keys()[group_dict.values().index(current_group)]).rstrip()
        group_class = [group_class_name,'Other']
        group_features = feature_names[:]
        single_y_index = 0
        while single_y_index < len(y_values):
            single_y_value = y_values[single_y_index]
            if single_y_value == current_group:
                group_y.append(1)
            else:
                group_y.append(2)
            single_y_index = single_y_index + 1

        while genes_in_group < num_genes_for_each_group:
            classifier = classifier.fit(group_x,group_y)
            graph_str = tree.export_graphviz(classifier, out_file=None,feature_names=group_features,class_names=group_class,filled=True,rounded=True,special_characters=True)
            sig_genes_dict = parse_top_node_of_graph_data(graph_str,group_class_name)
            for single_gene in sig_genes_dict:
                gene_index = group_features.index(single_gene)
                gene_values_dict = {}
                gene_avg_dict = {}
                single_cell_index = 0
                while single_cell_index < len(group_x):
                    cell_type = y_values[single_cell_index]
                    if cell_type not in gene_values_dict:
                        gene_values_dict[cell_type] = []
                    gene_values_dict[cell_type].append(group_x[single_cell_index][gene_index])
                    single_cell_index = single_cell_index + 1
                for single_cell_type in gene_values_dict:
                    gene_avg_dict[single_cell_type] = sum(gene_values_dict[single_cell_type])/float(len(gene_values_dict[single_cell_type]))
                gene_significant = True
                for single_mean_cell in gene_avg_dict:
                    if single_mean_cell != current_group:
                        if abs(gene_avg_dict[current_group] - gene_avg_dict[single_mean_cell]) < 20:
                            gene_significant = False
                group_features.remove(single_gene)
                for single_x_list in group_x:
                    single_x_list.pop(gene_index)
                if gene_significant == True:
                    significant_genes_avg[single_gene] = gene_avg_dict
                    if single_gene not in gene_list:
                        gene_list.append(single_gene)
                        gene_class.append(group_class_name)
                        genes_in_group = genes_in_group + 1
                    else:
                        gene_class_index = gene_list.index(single_gene)
                        gene_class[gene_class_index] = 'Multi: '+' + '.join([str_num for str_num in gene_class[gene_class_index] if str_num.isdigit()]) + ' + ' + str(current_group)
        current_group = current_group + 1
    gene_and_class_dict = {}
    current_gene_num = 0
    while current_gene_num < len(gene_list):
        gene_class_name = gene_class[current_gene_num]
        if gene_class_name not in gene_and_class_dict:
            gene_and_class_dict[gene_class_name] = []
        gene_and_class_dict[gene_class_name].append(gene_list[current_gene_num])
        current_gene_num = current_gene_num + 1
    group_names = []
    combined_group_names = []
    for single_class in gene_and_class_dict:
        if single_class.split()[0] == 'Group':
            group_names.append(single_class)
            gene_list = gene_and_class_dict[single_class]
            group_num = int(single_class.split()[1])
            new_sorted_list = []
            group_gene_list = []
            gene_score_list = []
            for single_gene in gene_list:
                avg_values = significant_genes_avg[single_gene]
                current_group_number = 1
                avg_score = 0
                while current_group_number <= len(avg_values):
                    if current_group_number != group_num:
                        avg_score = avg_score + avg_values[group_num] - avg_values[current_group_number]
                    current_group_number = current_group_number + 1
                group_gene_list.append(single_gene)
                gene_score_list.append(avg_score)
            combine_list = zip(gene_score_list,group_gene_list)
            new_sorted_score, new_sorted_genes = zip(*sorted(combine_list, reverse=True))
            sorted_genes_list = list(new_sorted_genes)
            gene_and_class_dict[single_class] = sorted_genes_list
        else:
            combined_group_names.append(single_class)
```

Figure D.1: Python code for heatmap generation

D.1.2 R

In the R script The significant genes, gene classes, cells, and cell class lists are loaded. Gene classes indicate what cell group the decision tree was able to isolate the best. The cells list indicate all the cells to be processed, and the cell class list indicates the group for each cell. The monocle dataset for classified cells is loaded and all the cells in the cells list are extracted from the dataset. In addition, all the genes indicated in the gene list are extracted from the expression matrix. The data is the annotated with gene class and cell type and sorted for both rows and cols. The matrix is then log transformed. One was added to each value to prevent negative value appearing for expression values less then 1. A heatmap is created from the data using pheatmap. The png is dynamically sized based on the size of the dataset and set to a resolution of 600 dpi. The png is then saved



```r
#Usage:
#Rscript cell_manual_differential.r gene_list gene_class cell_list cell_class
library("monocle")
library("reshape")
library("ggplot2")
library("pheatmap")
args <- commandArgs(trailingOnly=TRUE)
if(length(args)!=4)
{
        print("Usage: Rscript cell_manual_differential.r gene_list gene_class cell_list cell_class")
}
HSMM <- readRDS("classified_cells.rds")
HSMM <- HSMM[, strtoi(unlist(strsplit(args[3],",")))]
gene_list <-  unlist(strsplit(args[1],","))
gene_class <- unlist(strsplit(args[2],","))
cell_class <- unlist(strsplit(args[4],","))
saveRDS(gene_list,'gene_manual_list.rds')
saveRDS(gene_class,'gene_manual_class.rds')
newHSMM <- HSMM[gene_list,]
expression_matrix <- exprs(newHSMM)
colnames(expression_matrix) <- cell_class
sorted_matrix <- expression_matrix[order(gene_class),order(colnames(expression_matrix))]
Cell_Type <- colnames(sorted_matrix)
colnames(sorted_matrix) <- 1:length(colnames(sorted_matrix))
annotation_col_data <- data.frame(Cell_Type)
annotation_row_data <- data.frame(gene_class[order(gene_class)])
rownames(annotation_row_data) <- rownames(sorted_matrix)
names(annotation_row_data) <- "Gene Class"
names(annotation_col_data) <- "Cell Type"
log_matrix <- log(sorted_matrix+1)
png('cell_manual_heatmap.png',width = ceiling(length(rownames(annotation_col_data))/10), height = ceiling(length(rownames(annotation_row_data))/8), units= 'in',res=600)
pheatmap(log_matrix, annotation_col=annotation_col_data,annotation_row=annotation_row_data,cluster_cols = FALSE,cluster_rows = FALSE,show_colnames = FALSE)
dev.off()
```

Figure D.2: R code for heatmap generation

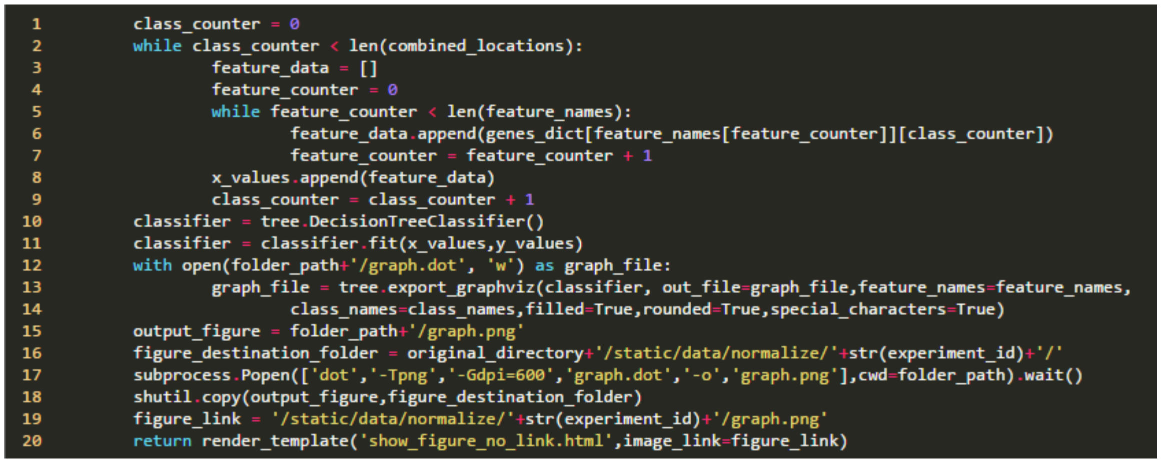
D.2 Decision Tree

The decision tree is used internally in the heatmap function. This function allows the users to visualize a decision tree generated from a given dataset. An external program is used to generate a 600 dpi png image of the tree. The image is then transfered into a folder where the user can access this image and a link is generated. The response is an html page with a image link to the figure.

D.3 Dynamic Form

D.3.1 Html

The html page for data input must contain dynamic elements in order to handle the variability in samples and conditions. The form starts with two conditions with two samples each. Under the two conditions, there are two buttons that allow the users to add or



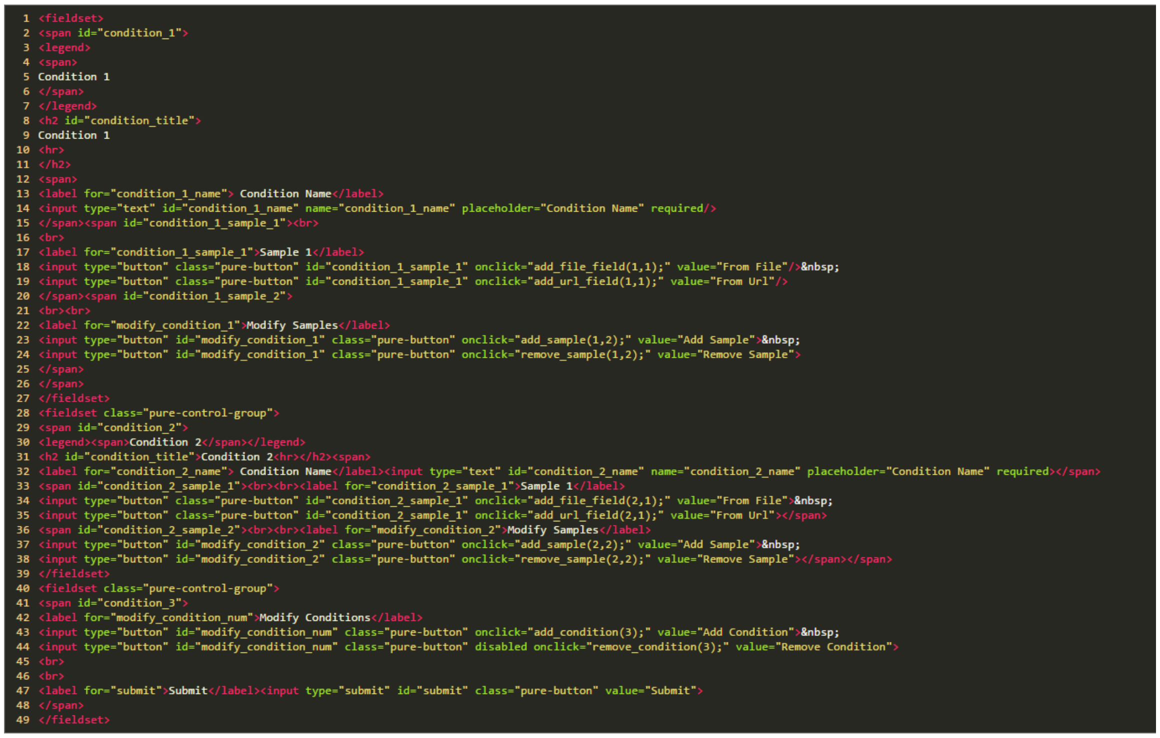
```python
class_counter = 0
while class_counter < len(combined_locations):
        feature_data = []
        feature_counter = 0
        while feature_counter < len(feature_names):
                feature_data.append(genes_dict[feature_names[feature_counter]][class_counter])
                feature_counter = feature_counter + 1
        x_values.append(feature_data)
        class_counter = class_counter + 1
classifier = tree.DecisionTreeClassifier()
classifier = classifier.fit(x_values,y_values)
with open(folder_path+'/graph.dot', 'w') as graph_file:
        graph_file = tree.export_graphviz(classifier, out_file=graph_file,feature_names=feature_names,
                class_names=class_names,filled=True,rounded=True,special_characters=True)
output_figure = folder_path+'/graph.png'
figure_destination_folder = original_directory+'/static/data/normalize/'+str(experiment_id)+'/'
subprocess.Popen(['dot','-Tpng','-Gdpi=600','graph.dot','-o','graph.png'],cwd=folder_path).wait()
shutil.copy(output_figure,figure_destination_folder)
figure_link = '/static/data/normalize/'+str(experiment_id)+'/graph.png'
return render_template('show_figure_no_link.html',image_link=figure_link)
```

Figure D.3: Python code for decision tree generation

remove conditions. In each sample, there are buttons linking to the javascript functions add_file_field and add_url_field. These javascript functions change the input type to a file or url respectively. The parameters represent the condition and sample number respectively. Underneath the two samples there are buttons linking to the javascript functions add_sample and remove_sample. These functions add and remove samples accordingly with the parameter scheme previously mentioned. There is another set of javascript functions add_condition and remove_condition which are located at the bottom of the form. These functions add and remove conditions from the form.

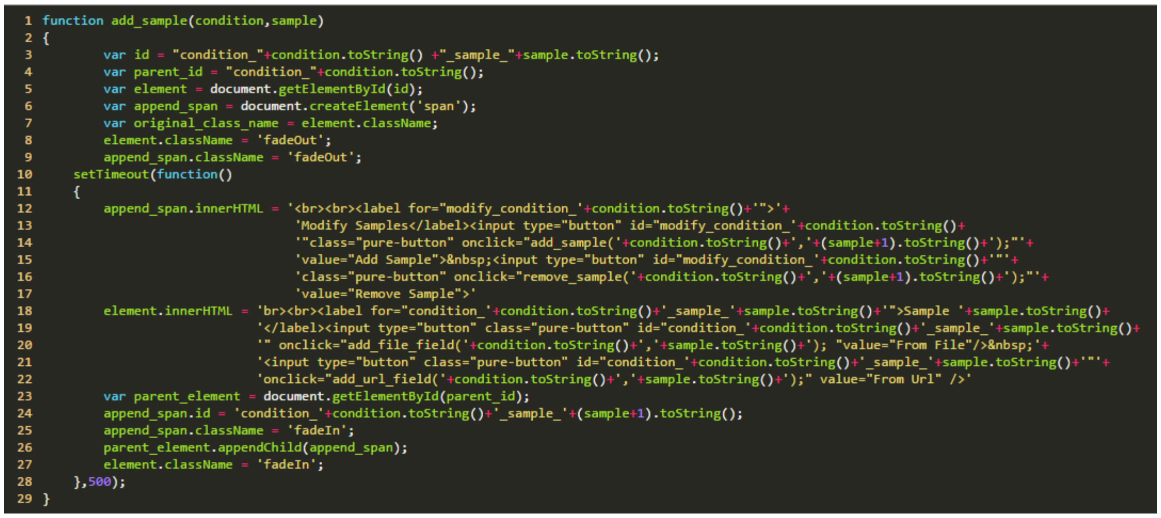
D.3.2 Javascript

The javascript functions are used to control the html form. The buttons call these functions and the javascript changes the html in the form and makes it dynamic. Only one javascript function is shown below, the add sample function. The function works by taking the parameters and finding the span with the appropriate id. New html is generated for the additional sample as well as a new button where the parameter is changed. The new html is added to the main html document after a timeout with the fadeIn className for visual effects.



```html
<fieldset>
<span id="condition_1">
<legend>
<span>
Condition 1
</span>
</legend>
<h2 id="condition_title">
Condition 1
<hr>
</h2>
<span>
<label for="condition_1_name"> Condition Name</label>
<input type="text" id="condition_1_name" name="condition_1_name" placeholder="Condition Name" required/>
</span><span id="condition_1_sample_1"><br>
<br>
<label for="condition_1_sample_1">Sample 1</label>
<input type="button" class="pure-button" id="condition_1_sample_1" onclick="add_file_field(1,1);" value="From File"/>&nbsp;
<input type="button" class="pure-button" id="condition_1_sample_1" onclick="add_url_field(1,1);" value="From Url"/>
</span><span id="condition_1_sample_2">
<br><br>
<label for="modify_condition_1">Modify Samples</label>
<input type="button" id="modify_condition_1" class="pure-button" onclick="add_sample(1,2);" value="Add Sample">&nbsp;
<input type="button" id="modify_condition_1" class="pure-button" onclick="remove_sample(1,2);" value="Remove Sample">
</span>
</span>
</fieldset>
<fieldset class="pure-control-group">
<span id="condition_2">
<legend><span>Condition 2</span></legend>
<h2 id="condition_title">Condition 2<hr></h2><span>
<label for="condition_2_name"> Condition Name</label><input type="text" id="condition_2_name" name="condition_2_name" placeholder="Condition Name" required></span>
<span id="condition_2_sample_1"><br><br><label for="condition_2_sample_1">Sample 1</label>
<input type="button" class="pure-button" id="condition_2_sample_1" onclick="add_file_field(2,1);" value="From File">&nbsp;
<input type="button" class="pure-button" id="condition_2_sample_1" onclick="add_url_field(2,1);" value="From Url"></span>
<span id="condition_2_sample_2"><br><br><label for="modify_condition_2">Modify Samples</label>
<input type="button" id="modify_condition_2" class="pure-button" onclick="add_sample(2,2);" value="Add Sample">&nbsp;
<input type="button" id="modify_condition_2" class="pure-button" onclick="remove_sample(2,2);" value="Remove Sample"></span></span>
</fieldset>
<fieldset class="pure-control-group">
<span id="condition_3">
<label for="modify_condition_num">Modify Conditions</label>
<input type="button" id="modify_condition_num" class="pure-button" onclick="add_condition(3);" value="Add Condition">&nbsp;
<input type="button" id="modify_condition_num" class="pure-button" disabled onclick="remove_condition(3);" value="Remove Condition">
<br>
<br>
<label for="submit">Submit</label><input type="submit" id="submit" class="pure-button" value="Submit">
</span>
</fieldset>
```

Figure D.4: HTML colde for a dynamic form



```javascript
function add_sample(condition,sample)
{
        var id = "condition_"+condition.toString() +"_sample_"+sample.toString();
        var parent_id = "condition_"+condition.toString();
        var element = document.getElementById(id);
        var append_span = document.createElement('span');
        var original_class_name = element.className;
        element.className = 'fadeOut';
        append_span.className = 'fadeOut';
    setTimeout(function()
    {
        append_span.innerHTML = '<br><br><label for="modify_condition_'+condition.toString()+'">'+
                                 'Modify Samples</label><input type="button" id="modify_condition_'+condition.toString()+
                                 '"class="pure-button" onclick="add_sample('+condition.toString()+','+(sample+1).toString()+');"'+
                                 'value="Add Sample">&nbsp;<input type="button" id="modify_condition_'+condition.toString()+'"'+
                                 'class="pure-button" onclick="remove_sample('+condition.toString()+','+(sample+1).toString()+');"'+
                                 'value="Remove Sample">'
        element.innerHTML = 'br><br><label for="condition_'+condition.toString()+'_sample_'+sample.toString()+'">Sample '+sample.toString()+
                            '</label><input type="button" class="pure-button" id="condition_'+condition.toString()+'_sample_'+sample.toString()+
                            '" onclick="add_file_field('+condition.toString()+','+sample.toString()+'); "value="From File"/>&nbsp;'+
                            '<input type="button" class="pure-button" id="condition_'+condition.toString()+'_sample_'+sample.toString()+'"'+
                            'onclick="add_url_field('+condition.toString()+','+sample.toString()+');" value="From Url" />'
        var parent_element = document.getElementById(parent_id);
        append_span.id = 'condition_'+condition.toString()+'_sample_'+(sample+1).toString();
        append_span.className = 'fadeIn';
        parent_element.appendChild(append_span);
        element.className = 'fadeIn';
    },500);
}
```

Figure D.5: Javascript code for a dynamic form

References

[1] Rogers, Michael P. "Working Linux into the CS curriculum." Journal of Computing Sciences in Colleges. Vol. 16. No. 1. Consortium for Computing Sciences in Colleges, 2000.

[2] Knobloch, Marlen. "The Role of Lipid Metabolism for Neural Stem Cell Regulation." Brain Plasticity Preprint: 1-12.

[3] Llorens-Bobadilla, Enric, et al. "Single-cell transcriptomics reveals a population of dormant neural stem cells that become activated upon brain injury." Cell Stem Cell 17.3 (2015): 329-340.

[4] Shin, Jaehoon, et al. "Single-cell RNA-seq with waterfall reveals molecular cascades underlying adult neurogenesis." Cell stem cell 17.3 (2015): 360-372.

[5] "Python Release Python 2.7.13." Python.org. N.p., n.d. Web. <https://www.python.org/downloads/release/python-2713/>.

[6] "Flask." Welcome — Flask (A Python Microframework). N.p., n.d. Web. <http://flask.pocoo.org/>.

[7] "JavaScript." Mozilla Developer Network. N.p., n.d. Web. <https://developer.mozilla.org/en-US/docs/Web/JavaScript>.

[8] "Fastq-dump." EdwardsLab. N.p., n.d. Web. <https://edwards.sdsu.edu/research/fastq-dump/>.

[9] "FastQC A Quality Control Tool for High Throughput Sequence Data." Babraham Bioinformatics. N.p., n.d. Web. <http://www.bioinformatics.babraham.ac.uk/projects/fastqc/>.

[10] "Open Source HTML5 Charts for Your Website." Chart.js. N.p., n.d. Web. <http://www.chartjs.org/>.

[11] Trapnell, Cole, Lior Pachter, and Steven L. Salzberg. "TopHat: discovering splice junctions with RNA-Seq." Bioinformatics 25.9 (2009): 1105-1111.

[12] Trapnell, Cole, et al. "Transcript assembly and quantification by RNA-Seq reveals unannotated transcripts and isoform switching during cell differentiation." Nature biotechnology 28.5 (2010): 511-515.

[13] Trapnell, Cole, et al. "Differential analysis of gene regulation at transcript resolution with RNA-seq." Nature biotechnology 31.1 (2013): 46-53.

[14] R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

[15] Trapnell, Cole, et al. ”The dynamics and regulators of cell fate decisions are revealed by pseudotemporal ordering of single cells.” Nature biotechnology 32.4 (2014): 381-386.

[16] Goff, L., C. Trapnell, and D. Kelley. ”cummeRbund: Analysis, exploration, manipulation, and visualization of Cufflinks high-throughput sequencing data.” R package version 2.0 (2013).

[17] Quinlan, J. Ross. ”Induction of decision trees.”Machine learning1.1 (1986): 81-106.

[18] Pedregosa, Fabian, et al. ”Scikit-learn: Machine learning in Python.” Journal of Machine Learning Research 12.Oct (2011): 2825-2830.

[19] Kolde, Raivo. ”Pheatmap: pretty heatmaps.” R package version 61 (2012).

[20] Dean, Jeffrey, and Sanjay Ghemawat. ”MapReduce: simplified data processing on large clusters.” Communications of the ACM 51.1 (2008): 107-113.

[21] Conesa, Ana, et al. ”A survey of best practices for RNA-seq data analysis.” Genome biology 17.1 (2016): 13.

[22] Wang, Zhong, Mark Gerstein, and Michael Snyder. ”RNA-Seq: a revolutionary tool for transcriptomics.” Nature reviews genetics 10.1 (2009): 57-63.