SLE712 ASSIGNMENT 3:

PRACTICAL 3 (BIOINFORMATICS) REPORT

# SLE712 - Bioinformatics and Molecular Biology Techniques

## MEGAN M. SORIA

## ID: 219389285

## ANKUSH DEHLIA

## ID: 219456078

### 29 MAY 2020

Contents

[SLE712 - Bioinformatics and Molecular Biology Techniques 1](#_Toc41221400)

[MEGAN M. SORIA 1](#_Toc41221401)

[ID: 219389285 1](#_Toc41221402)

[ANKUSH DEHLIA 1](#_Toc41221403)

[ID: 219456078 1](#_Toc41221404)

[29 MAY 2020 1](#_Toc41221405)

[1. Introduction 2](#_Toc41221406)

[1.1 Background (Bioiformatics) 2](#_Toc41221407)

[1.2 Project Details 2](#_Toc41221408)

[2. Part 1 2](#_Toc41221409)

[2.1 Overview 2](#_Toc41221410)

[2.2 Written answers for items 1 to 5 2](#_Toc41221411)

[2.3 Written answers for items 6 to 10 2](#_Toc41221412)

[3. Part 2 2](#_Toc41221413)

[3.1 Overview 2](#_Toc41221414)

[3.2 Written answers 2](#_Toc41221415)

# 1. Introduction

## 

## 1.1 Background (Bioinformatics)

Bioinformatics or computational biology is a branch of science that integrates biology and computer sciences. This interdisciplinary field is highly effective to analyze biological data using cutting-edge technology such as artificial intelligence, medical imaging, and genetic algorithm (Khan 2018). Moreover, it is used for statistical modeling, DNA sequence analysis, and gene expression analysis (Ayyildiz & Piazza 2019). Computer programming plays a vital role to interpret data and aids to process biological information. There many programming languages like Python, R, Java, Perl which are used by bioinformaticians (Bonnal et al. 2019). Among them, R is widely exploited for robust scripting.

R has several advantages over other programming languages which include open source, excellent visualization, vast package list, and wide syntax (Chan 2018). R can be run on Rstudio which is an integrated development environment (IDE) and free software. Rstudio can work on Mac, Windows and Linux operating systems or it can be operated online through the Rstudio cloud. One of the key features of the Rstudio is to work with projects that can be version controlled through git. Git track changes while working in a group and establishes coordination between programmers. Accessing a code through Github is a proficient way for software development and data mining. The aim of the practical is to develop skills in problem-solving, R coding, work together as a team using Rstudio and GitHub.

## 1.2 Project Details

**Github repository**: <https://github.com/megan0012/SLE712-Assignment-3.git>

# 2. Part 1

## 2.1 Overview

Importing files, data wrangling, mathematical operations, plots and saving code on GitHub

The file “gene\_expression.tsv” contains RNA-seq count data for two samples of interest.

The file “growth\_data.csv” contains measurements for tree circumference growing at two sites, control site and treatment site which were planted 20 years ago.

## 2.2 Written answers

**Que 1. Read in the file, making the gene accession numbers the row names. Show a table of values for the first six genes.**

Ans 1. The function read.csv() is used to read the file which is assigned to gene\_expression\_data. Inside the function read.csv, the gene accession numbers are designated as GeneID using an attribute row.names. Finally, the values of the first six genes are shown using the head() function and assigning n to 6.

gene\_expression\_data <- read.csv("Data/part1\_gene\_expression.tsv",

sep = '\t', row.names = "GeneID")

head(gene\_expression\_data, n=6)

**Que 2. Make a new column which is the mean of the other columns. Show a table of values for the first six genes.**

Ans 2. The new column for mean is created by using rowMeans() function which is assigned to gene\_expression\_data$Mean. $Mean with gene\_expression\_data adds a new column to the data. To show a table of values for the first six genes, head() function is applied with the attribute, 6.

gene\_expression\_data$Mean <- rowMeans(gene\_expression\_data[,1:2])

head(gene\_expression\_data, 6)

**Que 3. List the 10 genes with the highest mean expression.**

Ans 3. The order() function is applied to return data in ascending order. To get the highest mean expression, “-” is applied to gene\_expression\_data$Mean, which arranges the mean values from higher to lower. Additionally, the head() function is applied with attribute 10.

gene\_ordered <- gene\_expression\_data[order(-gene\_expression\_data$Mean),]

head(gene\_ordered, 10)

**Que 4. Determine the number of genes with a mean <10.**

Ans 4. The number of genes with a mean <10 were 43124. These were obtained by sub-setting gene\_expression\_data and taking the values of the mean using [,3]. By using function nrow(), the number of genes with a mean <10 was obtained.

mean\_lessthan10 <- gene\_expression\_data[(gene\_expression\_data[,3]<10),]

nrow(mean\_lessthan10) # 43124 genes

**Que 5. Make a histogram plot of the mean values in png format and paste it into your report.**

Ans 5. The histogram for the mean values was plotted using the function hist(). The argument xlab is used to label the x-axis and the main is used to create the title of the plot. In the histogram, the frequency is skewed to the right and does not give a proper picture of the data (Figure 1). Therefore, the sub-setting of the mean column was done and the extremes were taken out. However, before sub-setting, data were inspected and the ftable() function was used that gives the numerical data of the mean frequencies. The results obtained showed that the mean of the column was approximately 360, hence, extremes were removed which was less than 360 and greater than 2 (Figure 2). The images were saved as png format using the function png(). The png() creates a .png file and the argument filename can be used to input the filename and the location where the image is to be saved. The function dev.off() shuts down the png function, telling R that the file is finished and is ready to be saved. The plot function should be run in between png() and dev.off().

hist(gene\_expression\_data$Mean, xlab = "Mean", main = "Histogram of Gene Expression Data Mean")

ftable(gene\_expression\_data$Mean)

png(filename = "Data/part1\_histogram\_adjusted.png")

with(gene\_expression\_data, hist

(Mean[Mean>2 & Mean<360], breaks=seq(2,360,by=1),

xlab = "Mean", main = "Histogram of Gene Expression Data Mean>2 & Mean<360"))

dev.off()

A screenshot of a cell phone

Description automatically generated

Figure 1 Histogram of mean gene expression

A screenshot of a cell phone

Description automatically generated

Figure 2 Histogram of mean gene expression excluding extremes

**Que 6. Import this “growth\_data.csv” csv file into an R object. What are the column names?**

Ans 6. To import a csv file, the function read.csv() is used which is assigned to growth\_data. Inside the function, the header is taken as TRUE, which means the first row contains the column names. Moreover, attribute stringsAsFactors is assigned as FALSE, so that strings in the data cannot be considered as factors. When the function colnames() is applied to growth\_data, it returns the names of the column.

growth\_data <- read.csv("Data/part1\_growth\_data.csv",

header = TRUE, stringsAsFactors = FALSE)

column <- colnames(growth\_data)

**Que 7. Calculate the mean and standard deviation of tree circumference at the start and end of the study at both sites.**

Ans 7. To calculate the mean and standard deviation of tree circumference, mean() and sd() functions were applied, respectively. However, before calculating mean and standard deviation, the sub-setting of each site was performed. Sub-setting can be done by using the function subset(). The mean for the Northeast site at the start and the end of the study were calculated as 5.078 cm and 40.052 cm, respectively, while mean for the Southwest site at the start and the end of the study were calculated as 5.08 cm and 59.77 cm, respectively. The standard deviation for the Northeast site at the start and the end of the study were estimated as 1.06 and 16.90, respectively, while the standard deviation for the Southwest site at the start and the end of the study were calculated as 1.06 and 22.57, respectively.

*For Northeast*

ne <- subset(growth\_data, Site == "northeast")

head(ne)

*Mean for Northeast data*

mean\_end2 <- mean(ne$Circumf\_2004\_cm)

mean\_end1 <- mean(ne$Circumf\_2019\_cm)

mean\_end1

mean\_end2

*Standard Deviation for Northeast data*

sd(ne$Circumf\_2004\_cm)

sd(ne$Circumf\_2019\_cm)

*For Southwest*

sw <- subset(growth\_data, Site == "southwest")

head(sw)

*Mean for Southwest data*

mean\_start2 <- mean(sw$Circumf\_2004\_cm)

mean\_end2 <- mean(sw$Circumf\_2019\_cm)

mean\_start2

mean\_end2

*Standard Deviation for Southwest data*

sd(sw$Circumf\_2004\_cm)

sd(sw$Circumf\_2019\_cm)

**Que 8. Make a box plot of tree circumference at the start and end of the study at both sites.**

Ans 8. To make a boxplot from a set of values, the function boxplot() was applied. Arguments like names, ylab, and xlab were used to give names to each boxplot, label to the y-axis and x-axis, respectively. Furthermore, the attribute main was used to give a title to the plot and the col attribute was used to give color to the plots.

boxplot(ne$Circumf\_2004\_cm, ne$Circumf\_2019\_cm, sw$Circumf\_2004\_cm, sw$Circumf\_2019\_cm,

names = c("NE 2004", "NE 2019", "SW 2004","SW 2019"),

ylab= "Circumfrence (cm)" , xlab = "Sites and years" ,

main = "Growth at two different sites during 2004 and 2019", col= "green")

**Que 9. Calculate the mean growth over the past 10 years at each site.**

Ans 9. The mean growth over the past 10 years was calculated by subtracting the values of the circumference of trees in 2019 from the circumference of trees in 2009. Further, these values are assigned to a vector such as ne$growth and using the function mean(), the mean of the growth over the past 10 years were calculated. The mean growth for the Northeast site was calculated as 30.06 cm, while the mean growth for the Southwest site was estimated as 48.35 cm.

*Mean growth for Northeast data*

ne$growth <- (ne$Circumf\_2019\_cm - ne$Circumf\_2009\_cm)

mean\_growth\_ne <- mean(ne$growth)

mean\_growth\_ne

*Mean growth for Southwest data*

sw$growth <- (sw$Circumf\_2019\_cm - sw$Circumf\_2009\_cm)

mean\_growth\_sw<- mean(sw$growth)

mean\_growth\_sw

**Que 10. Use the t.test and wilcox.test functions to estimate the p-value that the 10 year growth is different at the two sites.**

Ans 10. T-test was performed to estimate the p-value by using the t.test() function. The p-value for the 10 year growth difference between the two site was calculated as 1.713e-06 which is equivalent to 0.000001713. For the Wilcox test, the function wilcox.test() was used. The p-value obtained through the Wilcox test was estimated as 4.626e-06 which is equivalent to 0.000004626.

*t test*

t\_test <- t.test(ne$growth,sw$growth)

t\_test

*Wilcox.test*

wilcox\_test <- wilcox.test(ne$growth, sw$growth)

wilcox\_test

## 

## 2.3 Written answers for items 6 to 10

# 3. Part 2

## 3.1 Overview

**Determine the limits of BLAST**

In this assignment we will be testing your ability to use supplied functions to perform an analysis into the limitsof BLAST. Your group will be allocated oneE. coli gene sequence found in the file: <https://raw.githubusercontent.com/markziemann/SLE712_files/master/bioinfo_asst3_part2_files/sample.fa> For example if your Rstudio username is student71 then your sequence is 71. Each group selects just 1sequence. Next, you will need the whole set of E. coli genes can be downloaded from this link: ftp://ftp.ensemblgenomes.org/pub/bacteria/release-42/fasta/bacteria\_0\_collection/escherichia\_coli\_str\_k\_12\_substr\_mg1655/cds/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa.gz

## 

## 3.2 Written answers

**Que 1. Download the whole set of *E. coli* gene DNA sequences and use gunzip to decompress. Use the makeblast() function to create a blast database. How many sequences are present in the E.coli set?**

Ans 1. The whole set of *E. coli* gene DNA sequence was downloaded using the function download.file(). Inside the function, the argument “destfile” can be used to specify the name and the destination folder to be used. To compress or decompress files with “.gzip” and “.bzip2” formats, the function gunzip() from the R.utils library can be used. The argument “overwrite” when set to FALSE does not remove the original compressed file after decompressing. Further, the makeblastdb() function from the rBLAST library was used to creates a BLAST database from a FASTA file. There were 4140 sequences present in the *E. coli* set.

download.file("ftp://ftp.ensemblgenomes.org/pub/bacteria/release-47/fasta/bacteria\_0\_collection/escherichia\_coli\_str\_k\_12\_substr\_mg1655/cds/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa.gz",

destfile = "Data/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa.gz")

R.utils::gunzip("Data/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa.gz", overwrite=FALSE)

rBLAST::makeblastdb("Data/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa",dbtype="nucl", "-parse\_seqids")

**Que 2. Download the sample FASTA sequences and read them in as above. For your allocated sequence, determine the length (in bp) and the proportion of GC bases.**

Ans 2. The sample FASTA sequences were downloaded by using the function download.file(). The read.fasta() function from the seqinr library was used to read the FASTA file and is assigned to sample\_fasta. Sequence 11 was taken from the sample\_fasta and is assigned to seq11 by sub-setting the sample\_fasta. The sequence length of seq11 was calculated by using the getLength() function from the seqinr library. There were 1497 base pairs in the seq11. Finally, the GC() function from the seqinr library was applied to sums all “G” and “C” bases from a seq11. The proportion of GC bases were computed to be 0.5744823.

download.file("https://raw.githubusercontent.com/markziemann/SLE712\_files/master/bioinfo\_asst3\_part2\_files/sample.fa", destfile = "Data/sample.fa")

sample\_fasta <- seqinr::read.fasta("Data/sample.fa")

Subset sequence 11

seq11 <- sample\_fasta[[11]]

Sequence length in bp

seqinr::getLength(seq11)

1497 bp

Proportion of GC bases

seqinr::GC(seq11)

0.5744823

**Que 3. You will be provided with R functions to create BLAST databases and perform blast searches. Use blast to identify what E. coli gene your sequence matches best. Show a table of the top 3 hits including percent identity, E-value and bit scores.**

Ans 3. The function myblastn\_tab() was provided to create BLAST databases and perform blast searches. The *E. coli* gene set was read by using read.fasta and is assigned to ecoli\_seq. BLAST search was performed using the provided function myblastn\_tab(). The arguments myseq can be used to assign sequence to be match while db specifies database to be used. The function was assigned to a variable results which returns the value of top hit. The percentage identity was found to be 100% with the given database. Further, top 3 hits were investigated and the results suggested that there were only one hit in the *E. coli* set. With 100% percentage identity, the E-value was calculated as 0 and the bitscore was computed as 2878.

ecoli\_seq <- seqinr::read.fasta("Data/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa")

results <- myblastn\_tab(myseq = seq11, db = "Data/Escherichia\_coli\_str\_k\_12\_substr\_mg1655.ASM584v2.cds.all.fa")

results

top3\_hits <- results[1:3,]

top3\_hits

qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue bitscore

1 11 AAC76604 100 1497 0 0 1 1497 1 1497 0 2878

NA NA <NA> NA NA NA NA NA NA NA NA NA NA

NA.1 NA <NA> NA NA NA NA NA NA NA NA NA NA

**Que 4. You will be provided with a function that enables you to make a set number of point mutations to your sequence of interest. Run the function and write an R code to check the number of mismatches between the original and mutated sequence.**

Ans 4. The function mutator() was provided to create a set number of point mutations in the sequence of interest. In a sequence 11, 100 mutations were made using the mutator() function. To make a pairwise alignment, seq11 was converted to a string using the c2s() function. The string of characters will then be converted into a DNAString object by using the DNAString() function. After conversion, the pairwiseAlignment() function from the Biostrings library was applied to check the number of mismatches between seq11 and seq11\_mut. The subject (in this case the mutated sequence) must be the second input of the function preceded by the pattern (in this case the original sequence) which can be a set of lists. The resulting alignment can now be used to determine the percent sequence identification using the pid() function. There was 94.92% sequence similarity between seq11 and seq11\_mut. To calculate number of mismatches between sequence 11 and mutated sequence 11, the function nmismatch() was applied. The results showed 76 number of mismatches between seq11 and seq11\_mut which could be due to overlapping of some base pairs.

create a mutated copy with 100 substitutions

seq11\_mut <- mutator(myseq=seq11,100)

now create a pairwise alignment

seq11\_mut\_ <- DNAString(c2s(seq11\_mut))

seq11\_ <- DNAString(c2s(seq11))

aln <- Biostrings::pairwiseAlignment(seq11\_,seq11\_mut\_)

pid(aln)

94.92318

nmismatch(aln)

76

**Que 5. Using the provided functions for mutating and BLASTing a sequence, determine the number and proportion of sites that need to be altered to prevent the BLAST search from matching the gene of origin. Because the mutation is random, you may need to run this test multiple times to get a reliable answer.**

Ans 5.

# 5. Using the provided functions for mutating and BLASTing a sequence, determine the number

# and proportion of sites that need to be altered to prevent the BLAST search from matching the

# gene of origin. Because the mutation is random, you may need to run this test multiple times

# to get a reliable answer.

# Write a fasta file and make a blast db from the traget sequence, seq11

write.fasta(seq11, names= "seq11", file.out = "Data/seq11.fa")

makeblastdb(file = "Data/seq11.fa", dbtype = "nucl")

# blast\_lim is a function that tests the maximum number of mutations that can still return

# a BLAST search match when compared to the original sequence. It takes an initial number of

# mutations used to mutate the original sequence, makes a BLAST search, and repeats this process in

# defined increments until the search returns NULL. It stores each iteration in a table with the

# last row as the highest number of mutations that returned a match. The following are the inputs:

# init\_mut initial number of mutations

# mut\_incr number of mutations added per iteration

blast\_lim <- function(init\_mut, mut\_incr){

# number of mutations

mut <- init\_mut

seq\_mut <- mutator(myseq=seq11,mut)

results <- myblastn\_tab(myseq = seq\_mut, db = "Data/seq11.fa")

# Save BLAST search results in a dataframe

results\_table <- as.data.frame(results)

# Insert new columnn for number of mutations

results\_table$num\_mut <- mut

# Keep mutating until BLAST search returns null

while (!is.null(results)){

# Number of added mutations per iteration

mut = mut + mut\_incr

seq\_mut <- mutator(myseq=seq11,mut)

results <- myblastn\_tab(myseq = seq\_mut, db = "Data/seq11.fa")

if (is.null(results)){ # Do not append the null search result to the table

results\_table

# Append search results and mutations if it is not empty

} else (results\_table[nrow(results\_table) + 1,] <- c(results,mut))

}

return(results\_table)

}

# Test the limits of BLAST search with different initial mutations

# and increments using the blast\_tester function

test1 <- blast\_lim(1,1)

test2 <- blast\_lim(1,10)

test3 <- blast\_lim(1,20)

test4 <- blast\_lim(1,30)

test5 <- blast\_lim(2,50)

# Merge all test results in one table and take the top 10 highest number of mutations

all\_tests <- rbind(test1, test2, test3, test4, test5)

max\_mut <- all\_tests[order(-all\_tests$num\_mut),]

head(max\_mut, 10)

# ANKUSH! no explanation what the previous function is used for, say here that it's weird 450 lim

# blast\_tester mutates a sequence in a defined number of places ("mut").

# If a BLAST search against the original sequence returns a match, the function returns a 1.

# If the search result is NULL, it returns a 0. Input:

# mut number of mutations to be applied in the sequence

blast\_tester <- function(mut){

seq\_mut <- mutator(myseq=seq11,mut)

results <- myblastn\_tab(myseq = seq\_mut, db = "Data/seq11.fa")

if (!is.null(results)){

return(1)

} else (return(0))

}

# Since the mutations are random, the BLAST search results changes in each run.

# The following code uses the results from the blast\_lim function and replicates the run

# of the blast\_tester function 100 times to get a mean value and a better grasp of the

# BLAST search behavior

# Create an empty data frame for blast\_tester function results

blast\_test\_res <- data.frame(matrix(ncol=2,nrow=0, dimnames=list(NULL, c("num\_of\_mut", "Mean\_blast\_res"))))

i <- 0

while (i < 800){

mean\_blast\_res <- mean(replicate(100,blast\_tester(i)))

blast\_test\_res[nrow(blast\_test\_res) + 1,] <- c(i,mean\_blast\_res)

i <- i + 50

}

**Que 6. Provide a chart or table that shows how the increasing proportion of mutated bases reduces the ability for BLAST to match the gene of origin. Summarize the results in 1 to 2 sentences.**

Ans 6.

A close up of a map

Description automatically generated

A close up of a map

Description automatically generated

blast\_test\_res$sites <- blast\_test\_res$num\_of\_mut

blast\_test\_res$prop <- blast\_test\_res$Mean\_blast\_res

blast\_test\_res$random <- blast\_test\_res$num\_of\_mut/1497

library("ggplot2")

ggplot(blast\_test\_res, aes(sites,prop)) +

geom\_line(color = "violet" ) +

geom\_point(shape=19,color="turquoise3", size= 2) +

theme\_bw() + labs(title="", x="Number of sites randomised",y="Proportion of successful BLASTs")

ggplot(blast\_test\_res, aes(random,prop)) +

geom\_line(color = "violet" ) +

geom\_point(shape=19,color="turquoise3", size= 2) +

theme\_bw()+ labs(title="", x="Number of sites randomised",y="Proportion of successful BLASTs")

## Bibliography

Ayyildiz, D & Piazza, S 2019, 'Introduction to Bioinformatics', *Methods Mol Biol*, vol. 1986, pp. 1-15.

Bonnal, RJP, Yates, A, Goto, N, Gautier, L, Willis, S, Fields, C, Katayama, T & Prins, P 2019, 'Sharing Programming Resources Between Bio\* Projects', *Methods Mol Biol*, vol. 1910, pp. 747-66.

Chan, BKC 2018, 'Data Analysis Using R Programming', *Adv Exp Med Biol*, vol. 1082, pp. 47-122.

Khan, NT 2018, 'Data Mining-Basics of Bioinformatics', *Transcriptomics: Open Access*, vol. 06, no. 01.