SLE712 ASSIGNMENT 3: PRACTICAL 3 (BIOINFORMATICS) REPORT

SLE712 - Bioinformatics and Molecular Biology Techniques

MEGAN M. SORIA
ID: 219389285

ANKUSH DEHLIA
ID: 219456078

JUNE 2020

Table of Contents

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 (1)

The project entails 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 (1)

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)
```

```
##                 SRR5150592 SRR5150593
## ENSG00000223972          1          0
## ENSG00000227232          0          1
## ENSG00000278267          0          0
## ENSG00000243485          0          0
## ENSG00000284332          0          0
## ENSG00000237613          0          0
```

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)
```

```
##                 SRR5150592 SRR5150593 Mean
## ENSG00000223972          1          0  0.5
## ENSG00000227232          0          1  0.5
## ENSG00000278267          0          0  0.0
## ENSG00000243485          0          0  0.0
## ENSG00000284332          0          0  0.0
## ENSG00000237613          0          0  0.0
```

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)
```

```
##                 SRR5150592 SRR5150593     Mean
## ENSG00000115414     311857     206347 259102.0
## ENSG00000210082     145916     163288 154602.0
## ENSG00000075624     133983     116762 125372.5
## ENSG00000198886      91596      99943  95769.5
## ENSG00000137801      95158      74546  84852.0
## ENSG00000198804      79832      84774  82303.0
## ENSG00000198786      74570      83589  79079.5
## ENSG00000196924      88225      66413  77319.0
## ENSG00000198712      76108      77108  76608.0
## ENSG00000108821      80342      60127  70234.5
```

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)
```

```
## [1] 43124
```

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).

```
hist(gene_expression_data$Mean, xlab = "Mean", main = "Histogram of Gene Expression Data Mean")
```

Figure 1 Gene Mean Values Histogram. The histogram for the Mean column produced only one bar graph near Mean = zero. This shows that the number of genes with mean =0 and mean < 500 far exceeds the number of genes with, approximately mean =< 500. The frequency is skewed to the right and does not give us a proper picture of the data.

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 to give 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().

```
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"))
```

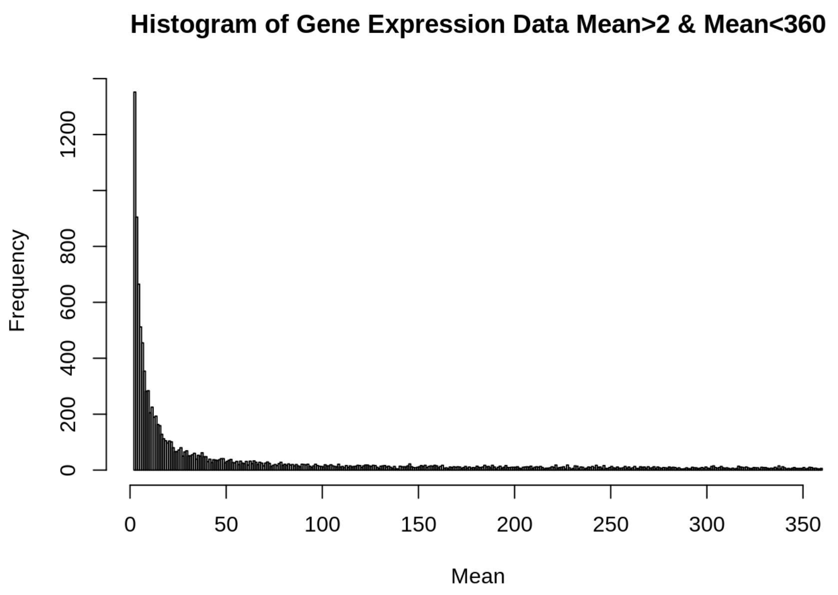


Figure 2 Gene Mean Values Histogram with subsetted data. Upon removal of extreme values, a different picture of the data is reflected in the histogram.

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)
column
```

```
## [1] "Site"            "TreeID"          "Circumf_2004_cm" "Circumf_2009_cm"
## [5] "Circumf_2014_cm" "Circumf_2019_cm"
```

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)
```

```
##        Site TreeID Circumf_2004_cm Circumf_2009_cm Circumf_2014_cm
## 1 northeast   A003             5.2            10.1            19.9
## 2 northeast   A005             4.9             9.6            18.9
## 3 northeast   A007             3.7             7.3            14.3
## 4 northeast   A008             3.8             6.5            10.9
## 5 northeast   A011             3.8             6.4            10.9
## 6 northeast   A012             5.9            10.0            16.8
##   Circumf_2019_cm
## 1            38.9
## 2            37.0
## 3            28.1
## 4            18.5
## 5            18.4
## 6            28.4
```

```
# Mean for Northeast data
mean_end2 <- mean(ne$Circumf_2004_cm)
mean_end1 <- mean(ne$Circumf_2019_cm)
mean_end1
```

```
## [1] 40.052
```

```
mean_end2
```

```
## [1] 5.078
```

```
# Standard Deviation for Northeast data
sd(ne$Circumf_2004_cm)
```

```
## [1] 1.059127
```

```
sd(ne$Circumf_2019_cm)
```

```
## [1] 16.90443
```

```
# For Southwest
sw <- subset(growth_data, Site == "southwest")
head(sw)
```

```
##         Site TreeID Circumf_2004_cm Circumf_2009_cm Circumf_2014_cm
## 51 southwest   A001             5.3            13.5            34.6
## 52 southwest   A002             5.2            10.1            19.8
## 53 southwest   A004             6.2            15.9            40.6
## 54 southwest   A006             5.1            11.5            25.9
## 55 southwest   A009             3.6             9.1            23.4
## 56 southwest   A010             6.6            14.9            33.6
##    Circumf_2019_cm
## 51            88.7
## 52            38.8
## 53           103.9
## 54            58.3
## 55            59.8
## 56            75.5
```

```
# Mean for Southwest data
mean_start2 <- mean(sw$Circumf_2004_cm)
mean_end2 <- mean(sw$Circumf_2019_cm)
mean_start2
```

```
## [1] 5.076
```

```
mean_end2
```

```
## [1] 59.772
```

```
# Standard Deviation for Southwest data
sd(sw$Circumf_2004_cm)
```

```
## [1] 1.060527
```

```
sd(sw$Circumf_2019_cm)
```

```
## [1] 22.57784
```

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")
```

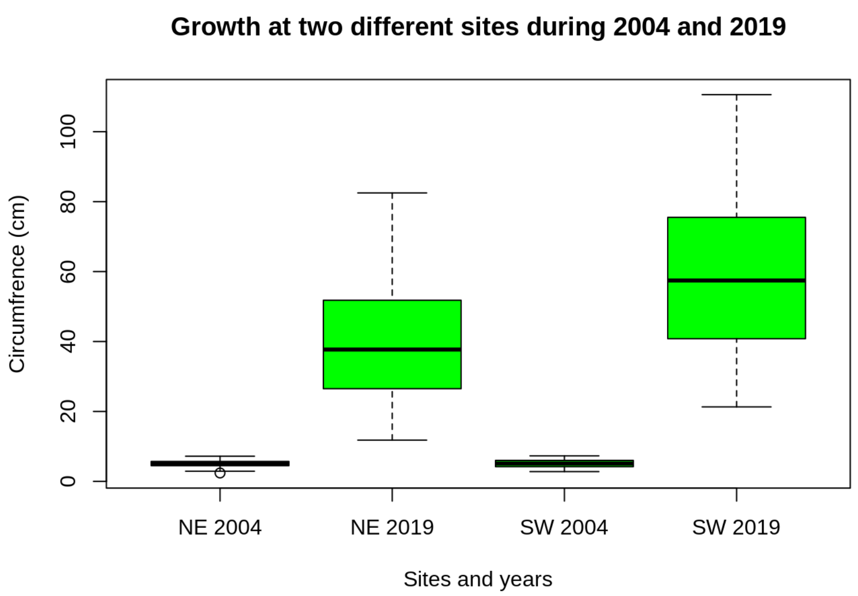


Figure 3 Box plot for tree circumference.

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
```

```
## [1] 30.076
```

```
# Mean growth for Southwest data

sw$growth <- (sw$Circumf_2019_cm - sw$Circumf_2009_cm)
mean_growth_sw<- mean(sw$growth)
mean_growth_sw
```

```
## [1] 48.354
```

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
```

```
##
##  Welch Two Sample t-test
##
## data:  ne$growth and sw$growth
## t = -5.124, df = 89.366, p-value = 1.713e-06
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -25.36543 -11.19057
## sample estimates:
## mean of x mean of y
##    30.076    48.354
```

```
# wilcox.test
wilcox_test <- wilcox.test(ne$growth, sw$growth)
wilcox_test
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  ne$growth and sw$growth
## W = 585, p-value = 4.626e-06
## alternative hypothesis: true location shift is not equal to 0
```

3. Part 2

3.1 Overview (2)

Determine the limits of BLAST

In this part of the project, supplied functions were used to perform an analysis into the limits of BLAST. An E. coli gene sequence found in the file:
https://raw.githubusercontent.com/markziemann/SLE712_files/master/bioinfo_asst3_part2_files/sample.fa (https://raw.githubusercontent.com/markziemann/SLE712_files/master/bioinfo_asst3_part2_files/sample.fa)

was allocated and Sequence 11 was used. A whole set of E. coli genes found in:
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.ASM584v
(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.ASM584v
was also used to create a blast database.

3.2 Prerequisite Libraries and Sources

Three R packages were used in part 2 of the project. `seqinr` is used to analyze sequence data. `R.utils` is used to extract compressed files. `rBLAST` is used as an interface to run BLAST searches. `ggplot2` is used to create and customize plots.
Two funtions were also used, found in the “source”, authored by Dr. Mark Ziemann.

```
library("seqinr")
library("R.utils")
```

```
## Loading required package: R.oo
```

```
## Loading required package: R.methodsS3
```

```
## R.methodsS3 v1.8.0 (2020-02-14 07:10:20 UTC) successfully loaded. See ?R.methodsS3 for help.
```

```
## R.oo v1.23.0 successfully loaded. See ?R.oo for help.
```

```
##
## Attaching package: 'R.oo'
```

```
## The following object is masked from 'package:R.methodsS3':
##
##     throw
```

```
## The following object is masked from 'package:seqinr':
##
##     getName
```

```
## The following objects are masked from 'package:methods':
##
##     getClasses, getMethods
```

```
## The following objects are masked from 'package:base':
##
##     attach, detach, load, save
```

```
## R.utils v2.9.2 successfully loaded. See ?R.utils for help.
```

```
##
## Attaching package: 'R.utils'
```

```
## The following object is masked from 'package:utils':
##
##     timestamp
```

```
## The following objects are masked from 'package:base':
##
##     cat, commandArgs, getOption, inherits, isOpen, nullfile, parse,
##     warnings
```

```
library("rBLAST")
```

```
## Loading required package: Biostrings
```

```
## Loading required package: BiocGenerics
```

```
## Loading required package: parallel
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following objects are masked from 'package:parallel':
##
##     clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,
##     clusterExport, clusterMap, parApply, parCapply, parLapply,
##     parLapplyLB, parRapply, parSapply, parSapplyLB
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, append, as.data.frame, basename, cbind, colnames,
##     dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,
##     grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,
##     order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,
##     rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,
##     union, unique, unsplit, which, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
## Loading required package: stats4
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:base':
##
##     expand.grid
```

```
## Loading required package: IRanges
```

```
##
## Attaching package: 'IRanges'
```

```
## The following object is masked from 'package:R.oo':
##
##     trim
```

```
## Loading required package: XVector
```

```
##
## Attaching package: 'Biostrings'
```

```
## The following object is masked from 'package:seqinr':
##
##     translate
```

```
## The following object is masked from 'package:base':
##
##     strsplit
```

```
library("ggplot2")
source("https://raw.githubusercontent.com/markziemann/SLE712_files/master/bioinfo_asst3_part2_files/mutblast_functions.R")
```

3.3 Written answers (2)

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.

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", destf
ile = "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)
```

```
## [1] 1497
```

```
# Proportion of GC bases
seqinr::GC(seq11)
```

```
## [1] 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.

```
myblastn_tab # provided R function
```

```
## function (myseq, db)
## {
##     mytmpfile1 <- tempfile()
##     mytmpfile2 <- tempfile()
##     write.fasta(myseq, names = attr(myseq, "name"), file.out = mytmpfile1)
##     system2(command = "/usr/bin/blastn", args = paste("-db ",
##         db, " -query", mytmpfile1, "-outfmt 6 -evalue 0.05 -ungapped >",
##         mytmpfile2))
##     res <- NULL
##     if (file.info(mytmpfile2)$size > 0) {
##         res <- read.csv(mytmpfile2, sep = "\t", header = FALSE)
##         colnames(res) <- c("qseqid", "sseqid", "pident", "length",
##             "mismatch", "gapopen", "qstart", "qend", "sstart",
##             "send", "evalue", "bitscore")
##     }
##     unlink(c(mytmpfile1, mytmpfile2))
##     if (!is.null(res)) {
##         res <- res[order(-res$bitscore), ]
##     }
##     res
## }
```

```
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
```

```
##   qseqid   sseqid pident length mismatch gapopen qstart qend sstart send evalue
## 1     11 AAC76604    100   1497        0       0      1 1497      1 1497      0
##   bitscore
## 1     2878
```

```
top3_hits <- results[1:3,]
top3_hits
```

```
##      qseqid   sseqid pident length mismatch gapopen qstart qend sstart send
## 1        11 AAC76604    100   1497        0       0      1 1497      1 1497
## NA       NA     <NA>     NA     NA       NA      NA     NA   NA     NA   NA
## NA.1     NA     <NA>     NA     NA       NA      NA     NA   NA     NA   NA
##      evalue bitscore
## 1         0     2878
## NA       NA       NA
## NA.1     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.

```
mutator # Provided R function
```

```
## function (myseq, nmut)
## {
##     myseq_mod <- myseq
##     mypos <- sample(seq_along(myseq), nmut)
##     myseq_mod[mypos] <- sample(c("a", "c", "g", "t"), length(mypos),
##         replace = TRUE)
##     return(myseq_mod)
## }
```

```
# 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)
```

```
## [1] 94.72278
```

```
nmismatch(aln)
```

```
## [1] 79
```

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: Two functions were created to answer this question. The first function `blast_lim` takes a sequence, mutates it using an initial (whole number) input, and increments the number of mutations in a new iteration until the search

gives a null result. This function was used in five diffrent tests using different initial mutations and increments. The results of all five tests were stored in a table. The top 10 highest number of mutation were subsetted. The values changes slightly for each run but the highest value is approximately under 400 mutations.

```
# 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 (whole number)
# mut_incr      number of mutations added per iteration (whole number)


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)
```

```
##     qseqid sseqid pident length mismatch gapopen qstart qend sstart send evalue
## 290     11  seq11 82.832   1497      257       0      1 1497      1 1497      0
## 282     11  seq11 82.766   1497      258       0      1 1497      1 1497      0
## 270     11  seq11 83.545   1495      246       0      1 1495      1 1495      0
## 253     11  seq11 84.729   1493      228       0      5 1497      5 1497      0
## 289     11  seq11 84.235   1497      236       0      1 1497      1 1497      0
## 252     11  seq11 85.101   1490      222       0      2 1491      2 1491      0
## 269     11  seq11 85.503   1490      216       0      2 1491      2 1491      0
## 281     11  seq11 85.608   1473      212       0     20 1492     20 1492      0
## 251     11  seq11 85.207   1494      221       0      2 1495      2 1495      0
## 250     11  seq11 85.638   1497      215       0      1 1497      1 1497      0
##     bitscore num_mut
## 290     1396     352
## 282     1390     331
## 270     1456     321
## 253     1556     311
## 289     1517     302
## 252     1584     301
## 269     1619     301
## 281     1609     301
## 251     1598     291
## 250     1638     281
```

A second approach was made using the second function, `blast_tester`, which takes an whole number as input as the number of mutations to be applied in the sequence. The function returns 1 if the search returns a result and a 0 if the search returns NULL. To account for randomness, this function is replicated 100 times and an average of the 1's and 0's to quantify the proportion of successful BLASTs. A while loop was used with an indicated an upper limit for the number of mutations (**800**). A fix interval of 50 was chosen to hasten the runtime of the function.

It can be noted that the initial suspected limit of **400** using the first function was incorrect. This is because the randomness factor was not taken into account. This was corrected by using 100 replications in the second approach.

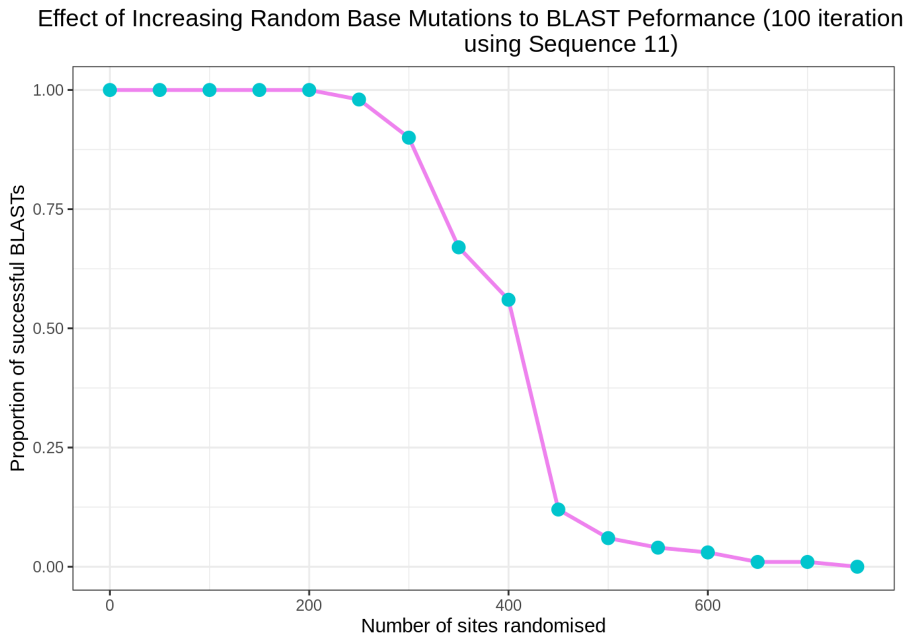
```
# blast_tester mutates a sequence in a defined number of places ("mut").
# If a BLAST search against the original sequence returns a match, the function returns a 1.
# If the search result is NULL, it returns a 0. Input:

# mut   number of mutations to be applied in the sequence (whole number)

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
}
blast_test_res
```

```
##    num_of_mut Mean_blast_res
## 1           0           1.00
## 2          50           1.00
## 3         100           1.00
## 4         150           1.00
## 5         200           1.00
## 6         250           0.98
## 7         300           0.90
## 8         350           0.67
## 9         400           0.56
## 10        450           0.12
## 11        500           0.06
## 12        550           0.04
## 13        600           0.03
## 14        650           0.01
## 15        700           0.01
## 16        750           0.00
```

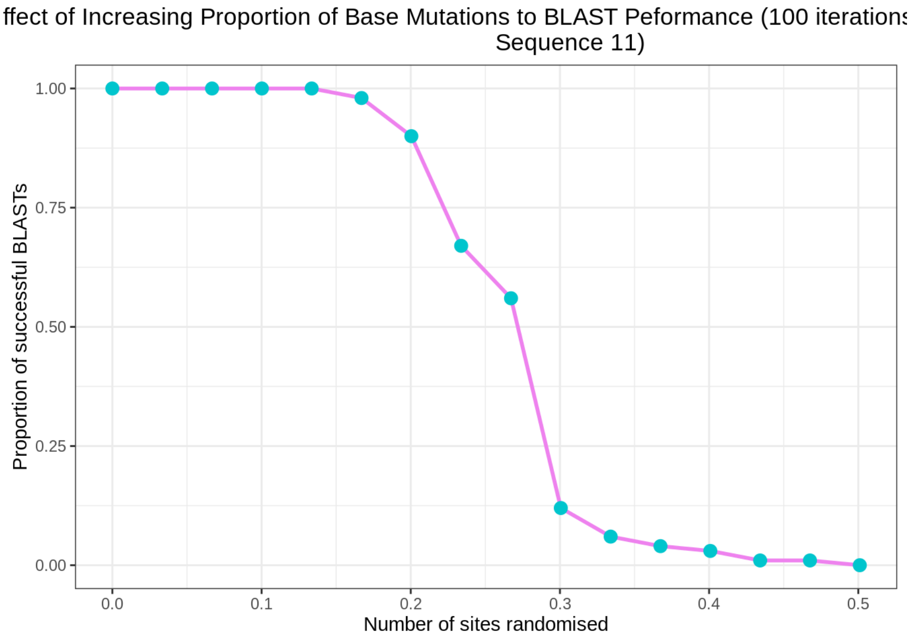
Que 6: Provide a chart or table that shows how the increasing proportion of mutated bases reduces the ability for BLAST to match the gene of origin. Summarize the results in 1 to 2 sentences.

Ans 6: For a sequence of 1497 bp, less than 43.42% or 650 maximum mutations are allowable to conduct a successful BLAST search. Mutations equal to or more than this number will yield a null result and can be concluded as the limit of a BLAST search.

```
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

ggplot(blast_test_res, aes(sites,prop)) +
        geom_line(color = "violet", size = 1 ) +
        geom_point(shape=19,color="turquoise3", size= 3) +
        theme_bw() + labs(title="Effect of Increasing Random Base Mutations to BLAST Peformance (100 iterations,
                          using Sequence 11)",
                          x="Number of sites randomised",
                          y="Proportion of successful BLASTs") + theme(plot.title = element_text(hjust = 0.5))
```




```
ggplot(blast_test_res, aes(random,prop)) +
        geom_line(color = "violet", size = 1 ) +
        geom_point(shape=19,color="turquoise3", size= 3) +
        theme_bw()+ labs(title="Effect of Increasing Proportion of Base Mutations to BLAST Peformance (100 iterations, using
                         Sequence 11)", x="Number of sites randomised",
                         y="Proportion of successful BLASTs") + theme(plot.title = element_text(hjust = 0.5))
```



4. References

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.