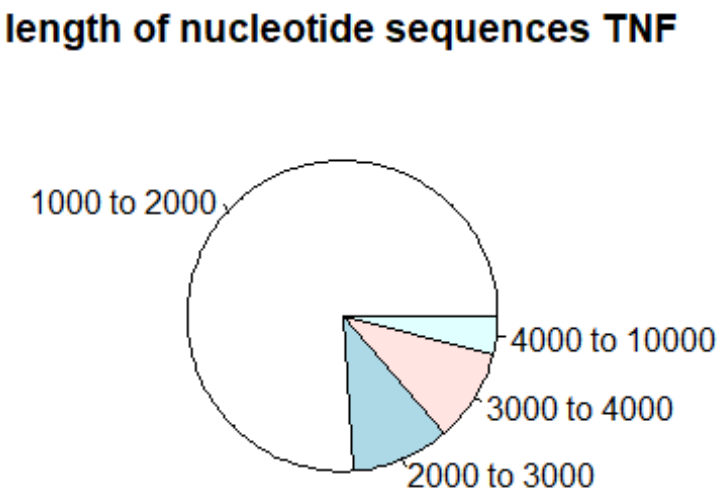
```
#Introduction

#Tumor necrosis factor (TNF) and tumor protein 53 (Tp53 or p53) are two of
the most widely studied and sequenced genes in eukaryotes, often due to
cancer studies. In a cancer patient, p53 mutation is often observed to
prevent the gene from regulating cell division in cancer cells (Hu et al.,
2021). Increased TNF expression in cancer cells also leads to augmented tumor
cell proliferation (Mercogliano et al., 2020).

#For this reason, the ability to distinguish the nucleotide sequence of a p53
gene from a mutated cancerous p53 gene as well as from other similar genes is
important in cancer research. It is therefore of great interest to develop a
machine learning model that can easily and reliably distinguish between these
two genes. This can then be expanded in future work to account for the
cancerous mutations of the P53 gene and allow for easier classification of
these genes in the future for new patients.


#initialize required variables
percentage = 0
i = 0
auc = 0
less_2000 <- vector()
less_3000 <- vector()
less_4000 <- vector()
less_10000 <- vector()
remove_TNF <- vector()
remove_P53 <- vector()

#load required libraries
library(rentrez)
library(seqinr)
library(Biostrings)
library(stringr)
library(naivebayes)
library(dplyr)
library(e1071)
library(ROCR)

#set working directory to desired location and ensure that it's at the
correct location
#setwd('D:/R/assignment_2')
#getwd()

#set an accuracy goal that you want the machine learning algorithm to achieve
(default is 90%)
goal <- 95

#our search of the NCBI database will query the nucleotide database for TNF
```

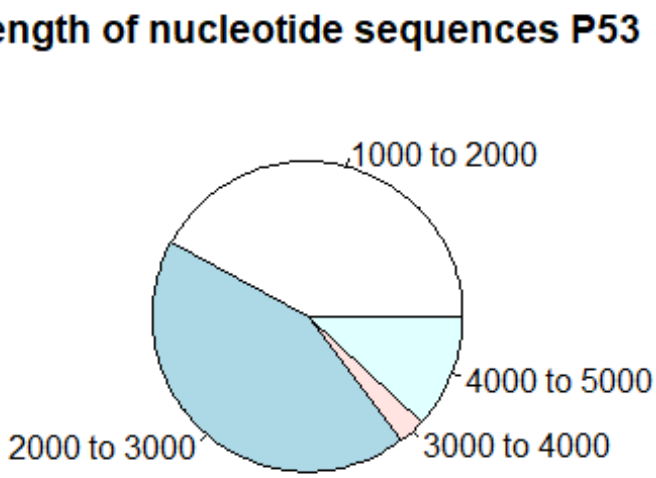
```
genes between 1000 and 10 000 base pairs long
NCBI_TNF <- entrez_search(db = "nuccore", term = "TNF[Gene Name] OR tnfa[Gene
Name] OR tnfb[Gene Name] AND 1000:10000[SLEN]")
NCBI_P53 <- entrez_search(db = "nuccore", term = "(P53[Gene Name]) OR
TP53[Gene Name] AND 1000:5000[SLEN] ")

#count the number of hits to modify the retmax value
new_retmax_TNF <- NCBI_TNF$count
new_retmax_P53 <- NCBI_P53$count

#Perform the search again with the new retmax value in order to obtain them
all. Because there are a lot of values, use_history is employed because the
result is too large and a sequence length between 1000 and 10000 was chosen
to avoid any whole genomes, further analysis will be done to assess if more
filtering needs to be done
NCBI_TNF <- entrez_search(db = "nuccore", term = "TNF[Gene Name] OR tnfa[Gene
Name] OR tnfb[Gene Name] AND 1000:10000[SLEN]", retmax = new_retmax_TNF,
use_history = TRUE)

NCBI_P53 <- entrez_search(db = "nuccore", term = "(P53[Gene Name]) OR
TP53[Gene Name] AND 1000:5000[SLEN]", retmax = new_retmax_P53, use_history =
TRUE)

#store the fasta sequences of our prior search
TNF_fetch <- entrez_fetch(db = "nuccore", rettype = "fasta", web_history =
NCBI_TNF$web_history)
P53_fetch <- entrez_fetch(db = "nuccore", rettype = "fasta", web_history =
NCBI_P53$web_history)

#we will write a fasta file to our working directory which will separate the
entries by new line
write(TNF_fetch, "TNF_fetch.fasta", sep = "\n")
write(P53_fetch, "P53_fetch.fasta", sep = "\n")

#read the fasta file as a DNAString set
TNF_stringset <- readDNAStringSet("TNF_fetch.fasta")
P53_stringset <- readDNAStringSet("P53_fetch.fasta")

#create a dataframe which holds the sequence as well as what they are and the
organism from which they were obtained
dataframe_TNF_original <- data.frame(source = names(TNF_stringset), sequence
= paste(TNF_stringset))
dataframe_P53_original <- data.frame(source = names(P53_stringset), sequence
= paste(P53_stringset))

#create a for loop which will subset each length difference of 1000
nucleotides into its own vector
for (k in 1:length(dataframe_TNF_original$sequence)){
  #each if statement checks if the number of characters that make up the
```

```
sample is less than a certain value and adds it to the appropriate vector
  if (nchar(dataframe_TNF_original$sequence[k]) < 2000) {less_2000 <-
append(less_2000, nchar(dataframe_TNF_original$sequence[k])) }
  else if (nchar(dataframe_TNF_original$sequence[k]) < 3000) { less_3000 <-
append(less_3000, nchar(dataframe_TNF_original$sequence[k])) }
  else if (nchar(dataframe_TNF_original$sequence[k]) < 4000) {less_4000 <-
append(less_4000, nchar(dataframe_TNF_original$sequence[k])) }
  else {less_10000 <- append(less_10000,
nchar(dataframe_TNF_original$sequence[k]))}
}

#pie_vector counts and stores the number of sequences in each group from the
for loop
pie_vector <-
c(length(less_2000),length(less_3000),length(less_4000),length(less_10000))
#pie_labels provides the labeling for the pie chart
pie_labels <- c("1000 to 2000", "2000 to 3000", "3000 to 4000", "4000 to
10000")
#produces the pie chart to see if any nucleotide lengths are underrepresented
and therefore should be excluded
pie(pie_vector, labels = pie_labels, main = "length of nucleotide sequences
TNF")
```




```
#reset the value of the vectors to use them again for the other gene
less_2000 <- vector()
less_3000 <- vector()
less_4000 <- vector()
```

```r
less_5000 <- vector()

#create a for loop which will subset each length difference of 1000
nucleotides into its own vector
for (k in 1:length(dataframe_P53_original$sequence)){
  #each if statement checks if the number of characters that make up the
sample is less than a certain value and adds it to the appropriate vector
  if (nchar(dataframe_P53_original$sequence[k]) < 2000) {less_2000 <-
append(less_2000, nchar(dataframe_P53_original$sequence[k])) }
  else if (nchar(dataframe_P53_original$sequence[k]) < 3000) { less_3000 <-
append(less_3000, nchar(dataframe_P53_original$sequence[k])) }
  else if (nchar(dataframe_P53_original$sequence[k]) < 4000) {less_4000 <-
append(less_4000, nchar(dataframe_P53_original$sequence[k])) }
  else {less_5000 <- append(less_5000,
nchar(dataframe_P53_original$sequence[k]))}
}

#pie_vector counts and stores the number of sequences in each group from the
for loop
pie_vector <-
c(length(less_2000),length(less_3000),length(less_4000),length(less_5000))
#pie_labels provides the labeling for the pie chart
pie_labels <- c("1000 to 2000", "2000 to 3000", "3000 to 4000", "4000 to
5000")
#produces the pie chart to see if any nucleotide lengths are underrepresented
and therefore should be excluded
pie(pie_vector, labels = pie_labels, main = "length of nucleotide sequences
P53")
```



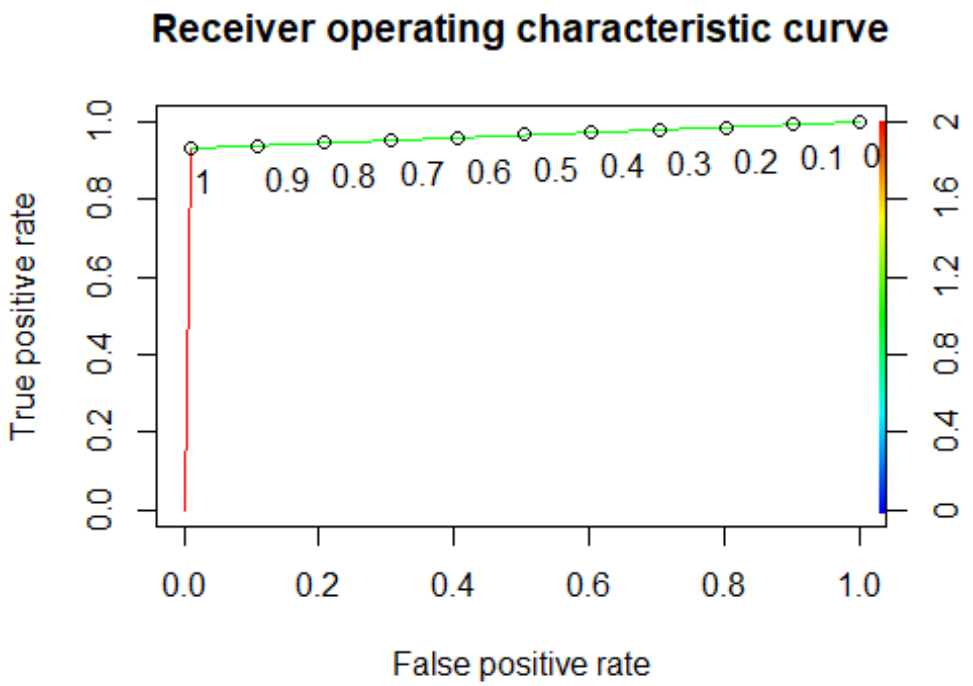

```
#looking at the data, the smaller end for both genes tends to have more
representation. This is logical as the incredibly long sequences ( >4000 for
TNF and >3000 for P53) are likely whole genomes or large parts of genomes
that contain the gene of interest among many other genes. It is therefore
wise to remove these samples and continue with only the sequence lengths that
are more likely to be the gene of interest.

#this for loop will create a list of the row number in which the TNF or P53
value are above the previously mentioned acceptable range
for (m in 1: length(dataframe_TNF_original$sequence)){
  if (nchar(dataframe_TNF_original$sequence[m]) > 4000) {remove_TNF <-
append(remove_TNF, m)}
}
for (n in 1: length(dataframe_P53_original$sequence)){
  if (nchar(dataframe_P53_original$sequence[n]) > 3000) {remove_P53 <-
append(remove_P53, n)}
}

#The remove list is used to generate a new dataframe that doesn't have the
undesired rows
dataframe_TNF_filtered <- dataframe_TNF_original[-remove_TNF, ]
dataframe_P53_filtered <- dataframe_P53_original[-remove_P53, ]

#confirm that the deletion worked and that the sequences with too high a
nucleotide count have been removed (the booleans should return FALSE)
unique(nchar(dataframe_TNF_filtered$sequence) > 4000)
```

```
## [1] FALSE

unique(nchar(dataframe_P53_filtered$sequence) > 3000)

## [1] FALSE

#change the object of type character (the sequence column) in the dataframe
to an object of type DNAstringset for future operations
dataframe_TNF_filtered$sequence <-
DNAStringSet(dataframe_TNF_filtered$sequence)
dataframe_P53_filtered$sequence <-
DNAStringSet(dataframe_P53_filtered$sequence)

#set the seed to a given value to ensure reproducibility
set.seed(420)

#remove unneeded variables and dataframes to free memory
rm(less_2000, less_3000, less_4000, less_5000, less_10000, remove_TNF,
remove_P53, NCBI_TNF, NCBI_P53, new_retmax_TNF, new_retmax_P53, TNF_fetch,
P53_fetch, TNF_stringset, P53_stringset, dataframe_TNF_original,
dataframe_P53_original, k, m, n, pie_labels, pie_vector)

#A while loop is used to change the length of the oligonucleotide that will
be used for the naive bayes algorithm. This will start at an oligonucleotide
length of 1 and increment until a
while (auc*100 < goal) {

  #i is incremented by one so that if the percentage hasn't achieved the
desired accuracy, it will be looped and increase the width of the
oligonucleotide to be used for the subsequent iteration
  i = i+1

  validation <- vector()

  #Create a copy of the original dataframe which will be reset during every
instance of the loop
  dataframe_TNF <- dataframe_TNF_filtered
  dataframe_P53 <- dataframe_P53_filtered

  #append the copy dataframe with the count of the oligonucleotides of length
i
  dataframe_TNF <- cbind(dataframe_TNF_filtered,
as.data.frame(oligonucleotideFrequency(dataframe_TNF$sequence, width = i)))
  dataframe_P53 <- cbind(dataframe_P53_filtered,
as.data.frame(oligonucleotideFrequency(dataframe_P53$sequence, width = i)))

  #convert the DNAStringset() object to that of character for further
operations
  dataframe_TNF$sequence <- as.character(dataframe_TNF$sequence)
  dataframe_P53$sequence <- as.character(dataframe_P53$sequence)
```

```
  #create another column in each dataframe to mark the dataframe as either
TNF or P53
  dataframe_TNF <- cbind(dataframe_TNF, data.frame(Gene_name = "TNF"))
  dataframe_P53 <- cbind(dataframe_P53, data.frame(Gene_name = "P53"))

  #set aside 20% of the overall data for validation ensuring an equal amount
of both TNF and P53 by choosing to take an amount equal to 20% of the
smaller set (in this case TNF)
  TNF_validation <- sample_n(dataframe_TNF, size = (0.2 *
length(dataframe_TNF$sequence)))
  P53_validation <- sample_n(dataframe_P53, size = (0.2 *
length(dataframe_TNF$sequence)))

  #merge both dataframes into a single dataframe (we can still distinguish
which sequence belongs to which gene due to the "Gene_name" column that   we
added earlier))
  validation_dataframe <- merge(TNF_validation,P53_validation, all.x = TRUE,
all.y = TRUE)

  #set aside the rest of the data for testing by filtering out the sequences
already found in the validation set
  TNF_training <- filter(dataframe_TNF,!dataframe_TNF$sequence %in%
TNF_validation$sequence)
  P53_training <- sample_n(filter(dataframe_P53,!dataframe_P53$sequence %in%
P53_validation$sequence),size = length(TNF_training$sequence))

  #merge both dataframes into a single dataframe (we can still distinguish
which sequence belongs to which gene due to the "Gene_name" column that   we
added earlier))
  training_dataframe <- merge(TNF_training,P53_training, all.x = TRUE, all.y
= TRUE)

  #perform the multinomial_naive_bayes analysis on the training dataframe.
The columns chosen will be from the third column (skipping "source and
sequence") to the second last column (which omits the last column
"gene_name")
  answer <- multinomial_naive_bayes(training_dataframe[,
3:(ncol(training_dataframe)-1)], training_dataframe$Gene_name)

  #The now trained naive bayes algorithm can then be used on the validation
set to predict the gene from which the sequence is taken. This will
result in a character vector
  prediction <- predict(answer, data.matrix(validation_dataframe[,
3:(ncol(training_dataframe)-1)]))

  #The character vector from the prediction and the "gene_name" column from
the validation set are taken and made into another data frame
  result <- data.frame(validation_dataframe[, ncol(validation_dataframe)],
```

```
prediction)

  #rename the column taken from the validation dataframe
  colnames(result)[1] <- "validation"

  #use a for loop to go line by line in the "results" dataframe in order to
create a numeric vector in which P53 is marked as 0 and TNF is marked    as 1
  for (j in 1:length(result$prediction)) {
    #if the row contains "P53", the validation vector will append a 0,
otherwise it will append a 1
    if (result$validation[j] == "P53")
    {
      validation <- append(validation, 0)
    }
    else
    {
     validation <- append(validation, 1)
    }
  }

  #convert the prediction to a vector of type "numeric" so it can be used to
calculate the AUC
  prediction <- as.numeric(prediction)
  #The prediction vector ranges from 1 to 2 but needs to be rearranged to
range from 0 to 1 instead
  prediction <- prediction-1
  #An ROC plot prediction is made using the prediction and validation numeric
vectors
  ROC_plot_prediction <- prediction(prediction, validation)
  #the area under the curve (auc) of the ROC plot is calculated to determine
the accuracy of the machine learning algorithm (0 means it's never
correct and 1 means it's always correct)
  auc <- as.numeric(performance(ROC_plot_prediction, "auc")@y.values)
}

## Warning: multinomial_naive_bayes(): x was coerced to matrix.

## Warning: multinomial_naive_bayes(): x was coerced to matrix.

## Warning: multinomial_naive_bayes(): x was coerced to matrix.

## Warning: multinomial_naive_bayes(): x was coerced to matrix.

## Warning: multinomial_naive_bayes(): x was coerced to matrix.

#Calculate the performance of the ROC prediction using "true positive
rate"(tpr) and "false positive rate"(fpr) as the variables
ROC_plot_performance <- performance(ROC_plot_prediction,"tpr", "fpr")
#plot the ROC plot performance
```

```
plot(ROC_plot_performance, colorize=TRUE, print.cutoffs.at=seq(0,1,by=0.1),
text.adj=c(-0.2,1.7), main = "Receiver operating characteristic curve")
```




```
#print out the final AUC score as a percentage accuracy as well as the final
width of the oligonucleotide
cat("with an oligonucleotide width of", i,  "a percentage accuracy of"
,auc*100, "was achieved.")
```

```
## with an oligonucleotide width of 5 a percentage accuracy of 96.1165 was
achieved.
```

```
#Results and discussion

#A naive bayes algorithm was chosen to solve this machine learning problem
due to its wide use and applicability. This is because this algorithm is
relatively fast and low on computing power as compared to other algorithms as
well as it not needing as large a training set to produce accurate
results(Kumbhar et al., 2013). This was useful in this case where the total
number of sets was approximately 1000. The pie charts showed that the NCBI
query contained a fair amount of unwanted data but excessive filtering would
also lead to an insufficient amount of sets to adequately train the model.
However, the filtering done appears to have been sufficient as the model is
able to achieve a high accuracy with only 4 oligonucleotides in sequence
needed.

#The Receiver operating characteristic curve (ROC) shows a sharp initial
increase which leads to a higher area under the curve (AUD) value. The AUD
```

gives the probability that a random sequence belonging to one of these two groups will be correctly classified so a high value indicates a well trained, accurate model(Al-Aidaroos et al., 2010). Future expansion on this work can include training the model to separate between other cancer related genes as well as the mutations of p53. This however is all dependant on there being enough sequencing data for these genes and mutations to adequately train the model.

#References

#Al-Aidaroos, K. M., Abu Bakar, A., & Othman, Z. (2010). Naïve Bayes variants in classification learning. Proceedings - 2010 International Conference on Information Retrieval and Knowledge Management: Exploring the Invisible World, CAMP'10, 276–281. https://doi.org/10.1109/INFRKM.2010.5466902

#Hu, J., Cao, J., Topatana, W., Juengpanich, S., Li, S., Zhang, B., Shen, J., Cai, L., Cai, X., & Chen, M. (2021). Targeting mutant p53 for cancer therapy: direct and indirect strategies. In Journal of Hematology and Oncology (Vol. 14, Issue 1). BioMed Central Ltd. https://doi.org/10.1186/s13045-021-01169-0

#Kumbhar, P., Jadhav, S. D., & Channe, H. P. (2013). Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques hemlata channe Related papers A Survey on Feat ure Select ion Techniques and Classificat ion Algorit hms for Efficient Text Cl… Comparative Study of K-NN, Naive Bayes and Decision Tree Classification Techniques. In International Journal of Science and Research (IJSR) ISSN (Vol. 5). www.ijsr.net

#Mercogliano, M. F., Bruni, S., Elizalde, P. v., & Schillaci, R. (2020). Tumor Necrosis Factor α Blockade: An Opportunity to Tackle Breast Cancer. In Frontiers in Oncology (Vol. 10). Frontiers Media S.A. https://doi.org/10.3389/fonc.2020.00584