

Project Report

SPAM DETECTION USING R

Prepared By:

Nikita Kunchanwar (800962459)

Nilanjan Chatterjee (800960960)

Priya Kumari (800964015)

Shashank Gupta (800970543)

Shruti Bilgundi (800963974)



ITCS 5102

Survey of Programming Language
University of North Carolina at Charlotte

TABLE OF CONTENT

S.No.	Topic
1.	Motivation
2.	Problem Description
3.	About the Language
4.	Elements of the Language
5.	Basic Control Abstractions of R
6.	Evaluation of R's writability, readability, and reliability:
7.	Major Strengths and Weaknesses of R
8.	Algorithm Used in Our Project
9.	Code
10.	Challenges Faced
11.	Conclusion
12.	Future Enhancements
13.	References

1. Motivation

Email has become one of the most important forms of communication. In 2014, there are estimated to be 4.1 billion email accounts worldwide, and about 196 billion emails are sent each day worldwide. Spam is one of the major threats posed to email users. In 2013, 69.6% of all email flows were spam. Links in spam emails may lead to users to websites with malware or phishing schemes, which can access and disrupt the receiver's computer system. These sites can also gather sensitive information from. Additionally, spam costs businesses around \$2000 per employee per year due to decreased productivity. Therefore, an effective spam filtering technology is a significant contribution to the sustainability of the cyberspace and to our society.

2. Problem description

We plan to build a spam detection engine using R where we can classify email/messages as spam or not spam. We have used the dataset from the UCI Machine Learning website <https://archive.ics.uci.edu/ml/datasets/Spambase>.

Our dataset consists of 57 attributes.

Of those first 48 attributes shows the count of certain single words in the email.

Next 6 attributes provides the count of characters such as ; ([! \$ #

The last 3 attributes consists the details of the use of capital letters in the email

These attributes about an email would be helpful for us to classify an email as SPAM or NOT SPAM.

Example : Consider an email with many counts of the word 'free' capitalized and many special characters.

"FREE!!! FREE!!! FREE!!!"

3. About the language

R is a language and environment for statistical computing and graphics. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.

R provides a wide variety of statistical (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) and graphical techniques, and is highly extensible. The S language is often the vehicle of choice for research in statistical methodology,

and R provides an Open Source route to participation in that activity.

One of R's strengths is the ease with which well-designed publication-quality plots can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.

R is available as Free Software under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.

The R environment:

R is an integrated suite of software facilities for data manipulation, calculation and graphical display. It includes

- an effective data handling and storage facility,
- a suite of operators for calculations on arrays, in particular matrices,
- a large, coherent, integrated collection of intermediate tools for data analysis,
- graphical facilities for data analysis and display either on-screen or on hardcopy, and
- a well-developed, simple and effective programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.

4. Elements of the language :

- **Reserved words :**

The following identifiers have a special meaning and cannot be used for object names - If, else, repeat, while, function, for, in, next, break, TRUE, FALSE, NULL, Inf, NaN, NA, NA_integer, NA_real, NA_complex, NA_character, etc.

- **Primitive data types :**

Numeric

Decimal values are called **numerics** in R. It is the default computational data type. If we assign a decimal value to a variable y as follows, y will be of numeric type

```
> y=12.5 # assign a decimal value
> y      # print the value of y
[1] 12.5
```

```
> class(y)
[1] "numeric" # print the class name of y
```

Furthermore, even if we assign an integer to a variable k, it is still being saved as a numeric value.

```
> k = 1
> k      # print the value of k
[1] 1
> class(k) # print the class name of k
[1] "numeric"
```

Integer

In order to create an integer variable in R, we invoke the `as.integer` function. We can be assured that y is indeed an integer by applying the `is.integer` function.

```
> y = as.integer(3)
> y      # print the value of y
[1] 3
> class(y) # print the class name of y
[1] "integer"
> is.integer(y) # is y an integer?
[1] TRUE
```

Incidentally, we can coerce a numeric value into an integer with the same `as.integer` function.

```
> as.integer(3.14) # coerce a numeric value
[1] 3
```

And we can parse a string for decimal values in much the same way.

```
> as.integer("5.27") # coerce a decimal string
[1] 5
```

On the other hand, it is erroneous trying to parse a non-decimal string.

```
> as.integer("Joe") # coerce an non-decimal string
[1] NA
Warning message:
NAs introduced by coercion
```

Often, it is useful to perform arithmetic on logical values. Like the C language, TRUE has the value 1, while FALSE has value 0.

```
> as.integer(TRUE) # the numeric value of TRUE
[1] 1
> as.integer(FALSE) # the numeric value of FALSE
[1] 0
```

Complex

A complex value in R is defined via the pure imaginary value i.

```
> z = 1 + 2i # create a complex number
> z          # print the value of z
[1] 1+2i
> class(z)   # print the class name of z
[1] "complex"
```

The following gives an error as -1 is not a complex value.

```
> sqrt(-1) # square root of -1
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
```

Instead, we have to use the complex value $-1 + 0i$.

```
> sqrt(-1+0i) # square root of -1+0i
[1] 0+1i
An alternative is to coerce -1 into a complex value.
> sqrt(as.complex(-1))
[1] 0+1i
```

Logical

A logical value is often created via comparison between variables.

```
> x = 1; y = 2 # sample values
> z = x > y    # is x larger than y?
> z           # print the logical value
[1] FALSE
> class(z)    # print the class name of z
[1] "logical"
```

Standard logical operations are "&" (and), "|" (or), and "!" (negation).

```
> u = TRUE; v = FALSE
> u & v        # u AND v
[1] FALSE
> u | v        # u OR v
[1] TRUE
> !u           # negation of u
[1] FALSE
```

Character

A character object is used to represent string values in R. We convert objects into character values with the `as.character()` function:

```
> x = as.character(3.14)
> x           # print the character string
[1] "3.14"
> class(x)    # print the class name of x
[1] "character"
```

Two character values can be concatenated with the `paste` function.

```
> fname = "Joe"; lname = "Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

However, it is often more convenient to create a readable string with the `sprintf` function, which has a C language syntax.

```
> sprintf("%s has %d dollars", "Sam", 100)
[1] "Sam has 100 dollars"
```

To extract a substring, we apply the `substr` function. Here is an example showing how to extract the substring between the third and twelfth positions in a string.

```
> substr("Mary has a little lamb.", start=3, stop=12)
[1] "ry has a l"
```

And to replace the first occurrence of the word "little" by another word "big" in the string, we apply the `sub` function.

```
> sub("little", "big", "Mary has a little lamb.")
[1] "Mary has a big lamb."
```

- **Structured types:** R has many data structures. These include

Atomic Vectors :

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. An atomic vector can be a vector of elements that are most commonly character, logical, integer or numeric.

You can create an empty vector with `vector()`. It is more common to use direct constructors such as `character()`, `numeric()`, etc.

```
x <- vector()
# with a length and type
vector("character", length = 10)
## [1] "" "" "" "" "" "" "" "" "" ""

character(5) ## character vector of length 5

## [1] "" "" "" "" ""
```



```
numeric(5)
```

```
## [1] 0 0 0 0 0
```

```
logical(5)
```

```
## [1] FALSE FALSE FALSE FALSE FALSE
```

Here is a vector containing three numeric values 2, 3 and 5.

```
> c(2, 3, 5)
```

```
[1] 2 3 5
```

And here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
```

```
[1] TRUE FALSE TRUE FALSE FALSE
```

A vector can contain character strings.

```
> c("aa", "bb", "cc", "dd", "ee")
```

```
[1] "aa" "bb" "cc" "dd" "ee"
```

Incidentally, the number of members in a vector is given by the length function.

```
> length(c("aa", "bb", "cc", "dd", "ee"))
```

```
[1] 5
```

Matrix

Matrices are a special vector in R. They are not a separate type of object but simply an atomic vector with dimensions added on to it. Matrices have rows and columns.

```
m <- matrix(nrow = 2, ncol = 2)
```

```
m
```

```
##      [,1] [,2]
## [1,]  NA  NA
## [2,]  NA  NA

dim(m)

## [1] 2 2
```

List

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types. Lists are sometimes called recursive vectors, because a list can contain other lists. This makes them fundamentally different from atomic vectors.

A list is a special type of vector. Each element can be a different type.

Create lists using `list()` or coerce other objects using `as.list()`

```
x <- list(1, "a", TRUE, 1 + (0+4i))
x

## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i

x <- 1:10
x <- as.list(x)
length(x)

## [1] 10
```

Factors

Factors are special vectors that represent categorical data. Factors can be ordered or unordered and are important when for modelling functions such as `lm()` and `glm()` and also in plot methods.

Factors can only contain pre-defined values. Factors can be created with `factor()`. Input is generally a character vector.

Factors can be created with `factor()`. Input is generally a character vector.

```
x <- factor(c("yes", "no", "no", "yes", "yes"))
x
## [1] yes no  no  yes yes
## Levels: no yes
```

`table(x)` will return a frequency table.

If you need to convert a factor to a character vector, simply use

```
as.character(x)

## [1] "yes" "no" "no" "yes" "yes"
```

Data frame

A data frame is a very important data type in R. It's pretty much the de facto data structure for most tabular data and what we use for statistics.

Data frames can have additional attributes such as `rownames()`, which can be useful for annotating data, like `subject_id` or `sample_id`. But most of the time they are not used.

Some additional information on data frames:

- Usually created by `read.csv()` and `read.table()`.
- Can convert to matrix with `data.matrix()`
- Coercion will be forced and not always what you expect.
- Can also create with `data.frame()` function.
- Find the number of rows and columns with `nrow(df)` and `ncol(df)`, respectively.
- Rownames are usually 1..n.

5 . Basic control abstractions of R :

- If statement: syntax - `if(condition=true) { cmd1 } else { cmd2 }`

Example -

```
x=5
if (x>2) {
  print(1)
}else{
  print(0)
}
```

Output : 1

- Ifelse statement: syntax - ifelse (condition, true_value, false_value)

Example -

```
x <- 1:11 # Creates sample data  
ifelse(x<4 | x>7, x, 0)
```

Output - 1 2 3 0 0 0 0 8 9 10 11

- While loop: syntax - while(condition) statement

Example -

```
z <- 0  
while(z < 10) {  
  z <- z + 2  
  print(z)  
}
```

Output - 2 4 6 8 10

- Repeat Loop statement:

Example -

```
z <- 0  
repeat {  
  z <- z + 1  
  print(z)  
  if(z > 100) break()  
}
```

Output - prints numbers from 1 to 101

Abstractions in R:

Functions: Syntax -

```
function.name <- function(arguments)  
{  
  //work with arguments  
}
```

```
computeSquareFunc<-function(x) {
  x*x
}
m<-10
#Call the function passing value of m
n<-computeSquareFunc(m)
```

Example-

Output - 100

Classes: R has three class systems which are S3, S4 and Reference class systems.

S3 -

```
s <- list(name = "Peter", age = 21)
class(s) <- "student"
s
```

Output -

```
$name
[1] "Peter"

$age
[1] 21

attr(,"class")
[1] "student"
```

S4 -

```
setClass("Person", representation(name = "character", age = "numeric"))
# create instance with new
peter <- new("Person", name = "Peter", age = 21)
```

Output -

```
An object of class "Person"
Slot "name":
[1] "Peter"

Slot "age":
[1] 21
```

Reference class -

```
setRefClass("student", fields = list(name = "character", age = "numeric"))
peter <- student(name = "Peter", age = 21)
peter
```

Output -

```
An object of class "Person"
Slot "name":
[1] "Peter"

Slot "age":
[1] 21
```

6. Evaluation of R's writability, readability, and reliability:

Writability: This is a measure of how easily a language can be used to develop programs. R can be expressed in a very clear and concise manner. R is not arbitrary in semantics and the language construct has minimal symbols which does not require many statements and focuses on simplification of code (concise).

Readability: R is a language and environment for statistical computing and graphics. The power of R lies in its libraries, and most of those have to do with either data visualization or statistics (or both). R syntax is quite readable and easy to learn but one should know how to run various statistical tests and read the output.

Reliability: The involvement of the scientific community in the development of R combined with the ever-growing engagement of large companies with R underlines the forcefulness with which R has conquered the market of data analytics in the last years, and raises expectations of an even brighter future.

7. Major Strengths and Weaknesses of R:

7.1 Strengths:

- **R is a Free, Open Source Language-** R is the open source equivalent of SAS, for what it's worth R can pretty much do everything SAS can do in terms of Statistical analysis and there are some pretty cool things R can do which SAS can't.
- **R is Cross-Platform Compatible-** One of the biggest advantages of R is that we can run R on several operating systems and different Software/Hardware. It is often used on Microsoft Windows (32-bit as well as 64-bit), Macintosh, GNU/Linux, UNIX and its derivatives such as Mac OS X, Darwin, FreeBSD, Solaris, etc. It also runs on some operating systems of Mainframes.
- **R easily Relates to other Programming Languages-** R can be attached to many file systems, applications and databases. Thus, R relates to other programming languages easily. Data could be easily pulled not only from Microsoft Excel, but also, Microsoft

Access, MySQL, SQLite, Oracle, and so on. Also, R can also be easily connected to various databases using ODBC (Open Database Connectivity Protocol) and the ROracle package.

- **High Graphical Capabilities:** Due to advanced graphical output, programmers love R. Various plots such as boxplots, histograms, polygons, scatterplots, barplots, arrows, trees, mathematical symbols, ggplots, all these are very impressive and self-explanatory. Thus, R is a fully programmable graphical language with outstanding features.

7.2 Weaknesses:

“R has a reputation of being hard to learn.”

- R has a steep learning curve. It does take awhile to get used to the power of R. Documentation is sometimes patchy and terse, and impenetrable to the non-statistician. However, some very high-standard books are increasingly plugging the documentation gaps.
- Knowing how to use various statistical tests is trickier. R is a statistical programming language and widely used to perform various tests like regression, clustering etc. One is required to have knowledge of statistics concepts to some extent to understand and interpret the output of the language. This knowledge is helpful in order to derive the best of the programming language.
- **Memory management, speed, and efficiency** are probably the biggest challenges that R faces. The design of the language can sometimes pose problems in working with very large data sets. Data has to be stored in physical memory. But as computers have more memory, this has become less of an issue.

8. Algorithm Used in Our Project:

We have used R implementation of **Support Vector Machine (SVM)** machine algorithm to predict if a given mail will be considered as spam or not spam.

SVM is a **classification algorithm** which means we will use it to predict if something belongs to a particular class. The objective of the SVM is to find **the optimal separating hyperplane which maximizes the margin of the training data**. Given a particular hyperplane, we can compute the **distance between the hyperplane and the closest data point**. Once we have this value, if we double it we will get what is called the **margin**. Support vector machines (SVMs) are a set of supervised learning methods used for classification, regression and outlier's detection. The advantages of support vector machines are that they are effective in high dimensional spaces

and still effective in cases where number of dimensions is greater than the number of samples. We have used two kernel values with five folds of c values for the run.

Implementation:

The above two parameters significantly influence the behavior of support vector machine.

C and accuracy : C refers to the penalty parameter term used for regularization. C parameter has no direct effect on the accuracy of the model in linearly-separable data sets. However, it can influence the accuracy of linearly inseparable models: for small values of C , we can expect high rates of error. As C increases, error rate gradually decreases and the change in error rate becomes almost insignificant after a particular value of C . The graph plots a monotonically and asymptotically decreasing curve.

Kernel points and hyper plane : Kernel points act as reference to classifying the training data on a hyperplane. This classification is coupled with the type of kernel chosen. For linearly separable data set, the linear model/kernel is most apt. In fact, the optical Character recognition assumes this kernel for modeling and does poorly with any other, particularly with rbf. For Spam data set, the kernel does not seem to have much of an influence on the accuracy. Any kernel like rbf or Laplacian is sufficiently efficient in modeling the given data set.

In our project, we are classifying email/messages as spam or not spam.

Packages we have used:

caret : Classification And Regression Training

It provides functions and tools for building classification and regression models

Some of the tools include:

- Data preparation and preprocessing
- Data splitting
- Model tuning and evaluation
- Variable importance estimation and selection

kernlab : Kernel-Based Machine Learning Lab

Kernel-based machine learning methods for classification, regression, clustering, novelty detection, quantile regression and dimensionality reduction. Among other methods 'kernlab' includes Support Vector Machines, Spectral Clustering, Kernel PCA, Gaussian Processes and a QP solver.

doParallel : Foreach Parallel Adaptor for the 'parallel' Package

As the name suggests, it helps in running parallel computations and reducing the running time.

9. Code:

```
#Explain the packages and functions used
```

```
#Consists of comma separated values
```

```
#spamdata.csv:
```

```
#Consists of column names
```

```
#spamnames.csv:
```

```
#Load the values
```

```
spamdata<- read.csv("spamdata.csv",header=FALSE,sep=";")
```

```
#Load the column names
```

```
spamnames<- read.csv("spamnames.csv",header=FALSE,sep=";")
```

```
#Set the column names of the dataframe spamdata
```

```
names(spamdata) <- sapply((1:nrow(spamnames)),function(i) toString(spamnames[i,1]))
```

```
#Converting into factors/categorical values so that we can use SVM for classification
```

```
spamdata$y <- factor(spamdata$y)
```

```
#Creating a sample dataset of 1000 rows
```

```
sample <- spamdata[sample(nrow(spamdata), 1000),]
```

```
#Loading the packages
```

```
#Loading caret
```

```
install.packages("caret", dependencies = c("Depends", "Suggests"))
```

```
require(caret)
```

```
#Loading kernlab
```

```
install.packages("kernlab", dependencies = c("Depends", "Suggests"))
```

```
require(kernlab)
```

```
#Loading doParallel
```

```
install.packages("doParallel", dependencies = c("Depends", "Suggests"))  
require(doParallel)
```

The doParallel package provides a parallel backend for the foreach/%dopar% function using the parallel package of R 2.14.0 and later.

```
#Split the data in two parts trainData and testData
```

```
trainIndex <- createDataPartition(sample$y, p = .8, list = FALSE, times = 1)  
trainData <- sample[ trainIndex,]  
testData <- sample[-trainIndex,]
```

Dataset is split into two parts- training data and test data. While splitting, 80% of data is moved to training data and remaining 20% constitute test data.

```
#set up multicore environment
```

```
registerDoParallel(cores=5)
```

The registerDoParallel function is used to register the parallel backend with the foreach package. Cores : The number of cores to use for parallel execution. If not specified, the number of cores is set to the value of options("cores"), if specified, or to one-half the number of cores detected by the parallel package.

```
# computing optimal value for the tuning parameter
```

```
sigDist <- sigDist(y ~ ., data = trainData, frac = 1)
```

Two parameters sigma and C are needed for training purpose. C is a parameter that controls the trade off between the achieving a low training error and a low testing error. A large C value gives us low bias and high variance and small C value gives us higher bias and lower variance.

```
svmTuneGrid <- data.frame(.sigma = sigDist[1], .C = 2^(-2:7))
```

train function in R fits predictive models over different tuning parameters. This can be used to tune models by picking the complexity parameters that are associated with the optimal resampling statistics. "method" argument specifies which classification model to use. 'preProc' argument is a string vector that defines a pre processing of predictor data. svmTuneGrid is assigned as tuneGrid which contains the possible tuning values. 'trControl' controls the

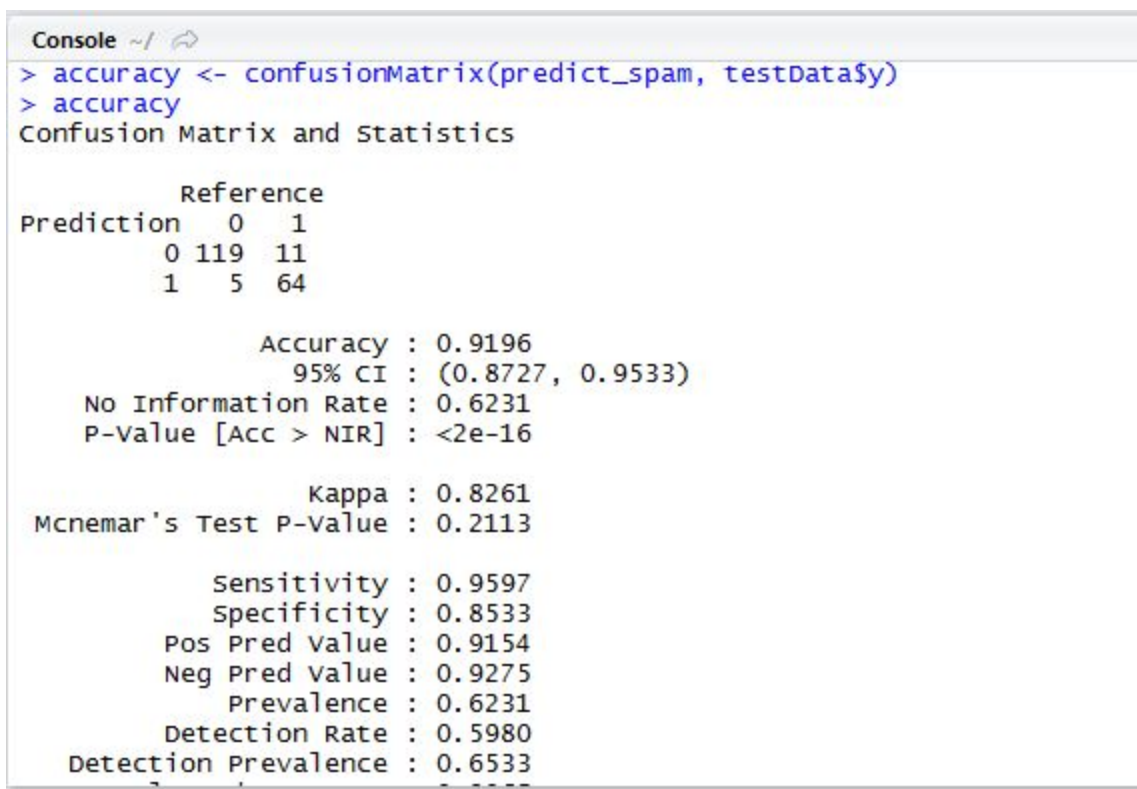
computations in the function. The resampling method used here is repeatedcv which does 5 repeats of 10-Fold Cross Validation for training data.

```
svm_model<- train(y ~ .,  
  data = trainData,  
  method = "svmRadial",  
  preProc = c("center", "scale"),  
  tuneGrid = svmTuneGrid,  
  trControl = trainControl(method = "repeatedcv", repeats = 5,  
    classProbs = FALSE))
```

#Evaluate the model

```
predict_spam <- predict(svm_model,testData[,1:57])
```

```
accuracy <- confusionMatrix(predict_spam, testData$y)
```



```
> accuracy <- confusionMatrix(predict_spam, testData$y)  
> accuracy  
Confusion Matrix and Statistics  
  
      Reference  
Prediction  0   1  
      0 119  11  
      1   5  64  
  
      Accuracy : 0.9196  
      95% CI : (0.8727, 0.9533)  
No Information Rate : 0.6231  
P-Value [Acc > NIR] : <2e-16  
  
      Kappa : 0.8261  
McNemar's Test P-Value : 0.2113  
  
      Sensitivity : 0.9597  
      Specificity : 0.8533  
Pos Pred Value : 0.9154  
Neg Pred Value : 0.9275  
Prevalence : 0.6231  
Detection Rate : 0.5980  
Detection Prevalence : 0.6533
```

#output file 'Result.csv' is written

```
write.csv(predict_spam, file = "Result.csv")
```

10. Challenges Faced:

- Analyze data and identify which machine learning algorithm is suitable for the problem statement.
- Learning new Machine learning algorithm and implementing it in R.
- Tweaking the model parameters to get good accuracy.

11. Conclusion:

- The project allowed us to classify mail as spam or not spam based on the given attributes such as word count frequency, special character frequency and the presence of capital letters in the mail. Using the Support Vector Machine (SVM) algorithm of R, we were able to get an accuracy above 90%.
- R was primarily developed for statistical analysis and is tough to learn for people with no statistical or mathematical background.
- However, R provides comprehensive packages and is a powerful programming language. With very less lines of code we were able to achieve our goal.

12. Future Enhancement:

- Other machine learning algorithm like logistic regression, decision trees, random forest algorithm can be implemented to build spam filtering model. The accuracy of the all the models could be compared for better understanding and result.
- The same model can also be created using other statistical tool such as SAS. This can help us to compare the various features of R and SAS.

13. References

<https://www.r-project.org/about.html>

<https://archive.ics.uci.edu/ml/datasets/Spambase>

<http://blog.revolutionanalytics.com/2014/04/seven-quick-facts-about-r.html>

<https://www.edureka.co/blog/why-learn-r/>

