Northeastern University
College *of* Professional Studies

Final Project Report

Dickson Wanjau

ALY6015 81058 Intermediate Analytics SPRING 2019 CPS

Instructor: Dr. Roseanna Hopper

Due Date: May 19, 2019.

Introduction

In today's society, social media has been a huge influence in our today's society and the way we interact with each other. We can stay in touch with our friends, associates and family members thousands of miles away. It has become a great avenue for meeting new friends and also for love interests. Internet dating has become more popular in this age than the traditional face-to-face meeting. We can market our products and pitch our entrepreneurial ideas online. We're kept abreast about what's happening all around the world at the touch of a button.

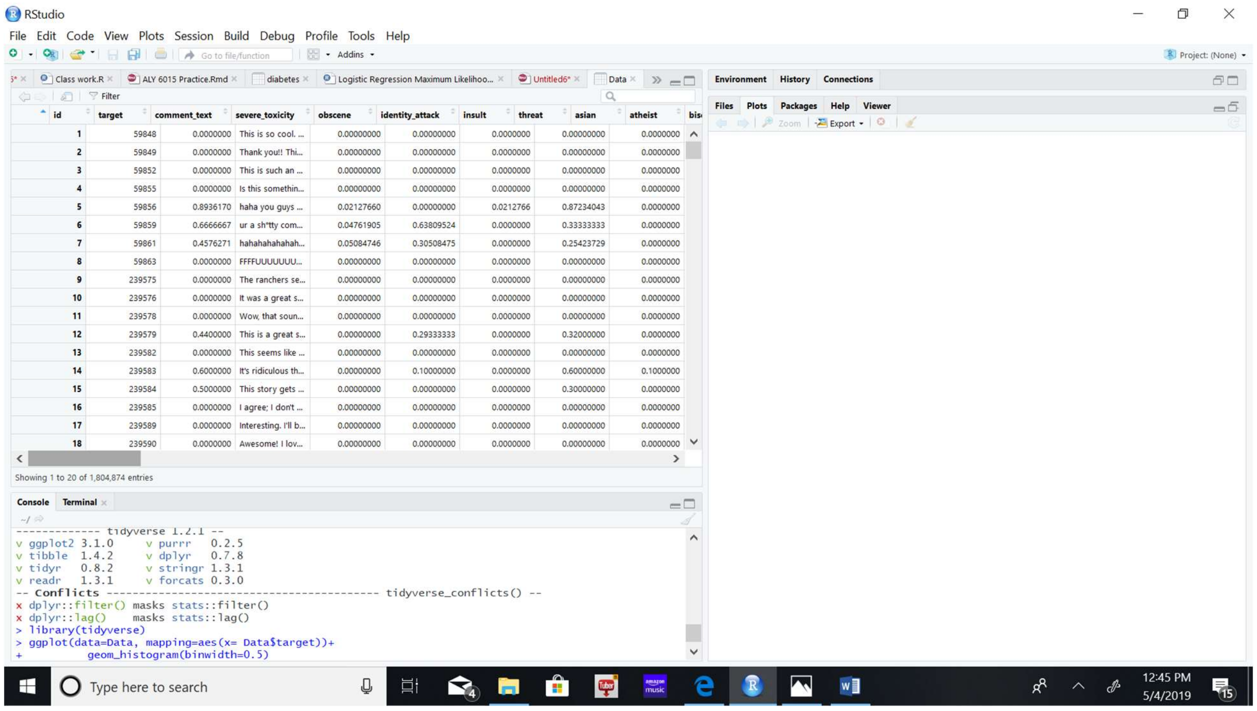
However, all these developments have brought many downsides such as phishing, identity theft, espionage but chief among these is cyber bullying. Criminals and other social media users can leave hurtful and intimidating messages on each other's profiles.

Analysis

Our study is based on a competition in Kaggle to help detect toxic comments in online conversations. The basic structure of our dataset after loading into R is as follows:

```
> library(bigmemory)
> str(Data)
'data.frame':	1804874 obs. of  45 variables:
 $ id                                 : int  59848 59849 59852 59855 59856 59
859 59861 59863 239575 239576 ...
 $ target                             : num  0 0 0 0 0.894 ...
 $ comment_text                       : Factor w/ 1780823 levels "\020Canada
is north of the USA border,  its colder in Canada.",..: 1460057 1268154 14605
75 779764 481606 1532685 482769 414436 1374638 823170 ...
 $ severe_toxicity                    : num  0 0 0 0 0.0213 ...
 $ obscene                            : num  0 0 0 0 0 ...
 $ identity_attack                    : num  0 0 0 0 0.0213 ...
 $ insult                             : num  0 0 0 0 0.872 ...
 $ threat                             : num  0 0 0 0 0 0 0 0 0 0 ...
 $ asian                              : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ atheist                            : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ bisexual                           : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ black                              : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ buddhist                           : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ christian                          : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ female                             : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ heterosexual                       : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ hindu                              : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ homosexual_gay_or_lesbian          : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ intellectual_or_learning_disability: num  NA NA NA NA 0.25 NA NA NA NA NA
...
 $ jewish                             : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ latino                             : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ male                               : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ muslim                             : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ other_disability                    : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ other_gender                        : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ other_race_or_ethnicity             : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ other_religion                      : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ other_sexual_orientation            : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ physical_disability                 : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ psychiatric_or_mental_illness       : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ transgender                         : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ white                               : num  NA NA NA NA 0 NA NA NA NA NA ...
 $ created_date                        : Factor w/ 1804362 levels "2015-09-29
10:50:41.987077+00",..: 1 2 3 4 5 6 7 8 173 174 ...
 $ publication_id                      : int  2 2 2 2 2 2 2 2 6 6 ...
 $ parent_id                           : num  NA NA NA NA NA ...
 $ article_id                          : int  2006 2006 2006 2006 2006 2006 20
06 2006 26662 26650 ...
 $ rating                              : Factor w/ 2 levels "approved","reject
ed": 2 2 2 2 2 2 2 2 1 1 ...
 $ funny                               : int  0 0 0 0 0 0 0 0 0 0 ...
 $ wow                                 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ sad                                 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ likes                               : int  0 0 0 0 1 0 0 0 0 1 ...
 $ disagree                            : int  0 0 0 0 0 0 0 0 0 0 ...
 $ sexual_explicit                     : num  0 0 0 0 0 ...
 $ identity_annotator_count            : int  0 0 0 0 4 0 0 0 0 0 ...
 $ toxicity_annotator_count            : int  4 4 4 4 47 105 59 4 4 4 ...
```

Our dataset contains 1,804,874 observations (rows) and 48 variables. This is a sneek peek of our data frame.



	id	target	comment_text	severe_toxicity	obscene	identity_attack	insult	threat
1	59848	0.0000000	This is so cool. ...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
2	59849	0.0000000	Thank you!! Thi...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
3	59852	0.0000000	This is such an ...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
4	59855	0.0000000	Is this somethin...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
5	59856	0.8936170	haha you guys ...	0.02127660	0.00000000	0.0212766	0.87234043	0.0000000
6	59859	0.6666667	ur a sh*tty com...	0.04761905	0.63809524	0.0000000	0.33333333	0.0000000
7	59861	0.4576271	hahahahahahah...	0.05084746	0.30508475	0.0000000	0.25423729	0.0000000
8	59863	0.0000000	FFFFUUUUUUU...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
9	239575	0.0000000	The ranchers se...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
10	239576	0.0000000	It was a great s...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
11	239578	0.0000000	Wow, that soun...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
12	239579	0.4400000	This is a great s...	0.00000000	0.29333333	0.0000000	0.32000000	0.0000000
13	239582	0.0000000	This seems like ...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
14	239583	0.6000000	It's ridiculous th...	0.00000000	0.10000000	0.0000000	0.60000000	0.1000000
15	239584	0.5000000	This story gets ...	0.00000000	0.00000000	0.0000000	0.30000000	0.0000000
16	239585	0.0000000	I agree; I don't ...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
17	239589	0.0000000	Interesting. I'll b...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000
18	239590	0.0000000	Awesome! I lov...	0.00000000	0.00000000	0.0000000	0.00000000	0.0000000

Showing 1 to 20 of 1,804,874 entries

```
------------- tidyverse 1.2.1 --
v ggplot2 3.1.0     v purrr   0.2.5
v tibble  1.4.2     v dplyr   0.7.8
v tidyr   0.8.2     v stringr 1.3.1
v readr   1.3.1     v forcats 0.3.0
-- Conflicts ---------------------------------------- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
> library(tidyverse)
> ggplot(data=Data, mapping=aes(x= Data$target))+
+         geom_histogram(binwidth=0.5)
```

As we notice, all our variables, except for the "comment_text" and "created_date", are factor (string) variables which were converted into numeric variables which we require for use in any statistical and machine learning algorithms. They are converted to binary dummy variables in a process called one hot encoding.

The dataset is labeled for identity mentions. For example if an obscenity is present in the comment, it's weighted on a scale of 0 to 1 e.g. 0.294. The cumulative total for all the variables is stored in the "target" variable.

For example, let's consider the following comment:

Comment: *i'm a white woman in my late 60's and believe me, they are not too crazy about me either!!*

Toxicity Labels: All 0.0

Identity Mention Labels: female: 1.0, white: 1.0 (all others 0.0)

We are interested in building a classification models on the "target" variable. We are tasked to first built toxicity models associate the names of frequently attacked identities with toxicity.

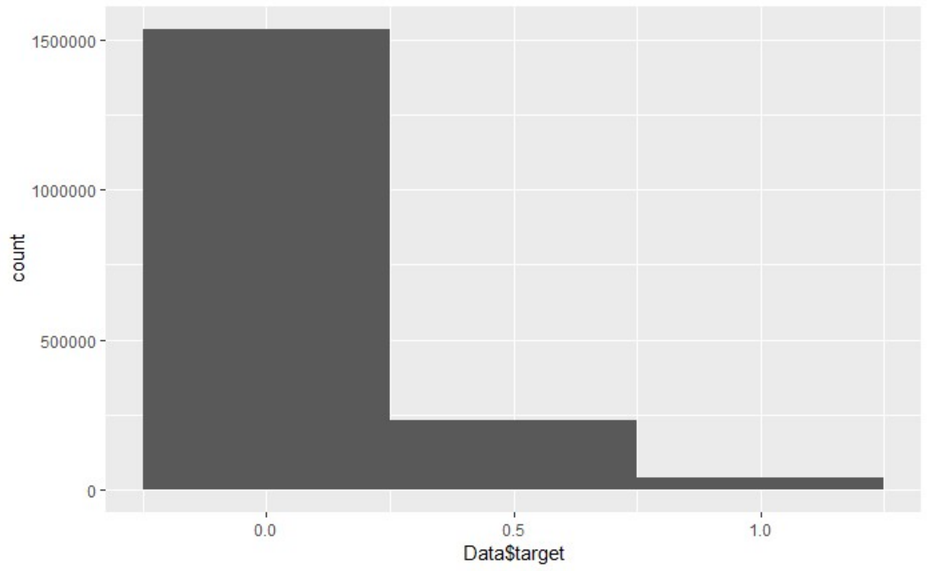
Using the 0.5 threshold, we impute the target vector into toxic and non-toxicity as follows:

```
> Data[which(Data$target>0.5),2]<-"Toxic"
> Data[which(Data$target<=0.5),2]<-"Non-Toxic"
> table(Data$target)

Non-Toxic     Toxic
  1698436    106438
```

We can build a histogram on our target variable to enable us visualize its distribution as follows:

```
library(tidyverse)
> ggplot(data=Data, mapping=aes(x= Data$target))+
+          geom_histogram(binwidth=0.5)
>
```



As we can see, the distribution is positively skewed with majority of the values leaning towards 0 (“non-toxicity”).

Data Preparation

Our data contains messages that are strings of text composed of words, spaces, numbers, and punctuation. Handling this type of complex data takes a large amount of thought and effort. We need to consider how to remove numbers, punctuation, handle uninteresting words such as “and”, “but”, and “or”, and how to break apart sentences into individual words.

Thankfully, this functionality has been provided by members of the R community in a text mining package titled “tm”.

```
> library(tm)
package �tm� was built under R version 3.5.3Loading required package: NLP
> data_corpus <- Corpus(VectorSource(Data$comment_text))
```

The first step in processing text data involves creating a corpus, which refers to a collection of text documents.

This command uses two functions. First, the Corpus() function creates an R object to store text documents. This function takes a parameter specifying the format of the text documents to be loaded. Since we have already read the comment messages and stored them in an R vector, we specify VectorSource(), which tells Corpus() to use the messages in the vector data $comment_text.

To look at the contents of the corpus, we can use the inspect() function. By combining this with methods for accessing vectors, we can view specific SMS messages, for instance:

```
> inspect(data_corpus[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] This is so cool. It's like, 'would you want your mother to read this??' R
eally great idea, well done!
[2] Thank you!! This would make my life a lot less anxiety-inducing. Keep it
up, and don't let anyone get in your way!
[3] This is such an urgent design problem; kudos to you for taking it on. Ver
y impressive!
```

The function tm_map() provides a method for transforming (that is, mapping) a tm corpus. We will use this to clean up our corpus using a series of transformation functions, and save the result in a new object called “corpus_clean”.

```
> corpus_clean <- tm_map(data_corpus, tolower)# converts all comments to lowe
r case
> corpus_clean <- tm_map(corpus_clean, removeNumbers) # removes all numbers i
n the comment
```

A common practice when analyzing text data is to remove filler words such as “to”, “and”, “but”, and “or”. These are known as stop words. We use the stopwords() function provided by the “tm” package.

```
>  corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())# remove
stop words
```

We'll also remove punctuations:

```
> corpus_clean <- tm_map(corpus_clean, removePunctuation)# removes punctuatio
n
```

Now the text messages are left with blank spaces where these characters used to be. The last step then is to remove additional whitespace, leaving only a single space between words.

```
> corpus_clean <- tm_map(corpus_clean, stripWhitespace)# removes white spaces
```

Now we can view the first three comments after the cleanup process.

```
> inspect(corpus_clean[1:3])#Inspect the first 3 messages after the clean up
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1]  cool like want mother read really great idea well done
[2] thank make life lot less anxietyinducing keep let anyone get way
[3]  urgent design problem kudos taking impressive
```

The final step is to split the messages into individual components through a process called tokenization. A token is a single element of a text string.

The DocumentTermMatrix() function will take the corpus and create a data structure called a sparse matrix, in which the rows of the matrix indicate documents (that is, comment texts) and the columns indicate terms (that is, words).

```
> sms_dtm <- DocumentTermMatrix(corpus_clean)
```

Each cell in the matrix stores a number indicating a count of the times the word indicated by the column appears in the document indicated by the row

This will tokenize the corpus and return the sparse matrix with the name “sms_dtm”. From here, we'll be able to perform analyses involving word frequency.

Creating training and test datasets

Since our data have been prepared for analysis, we now need to split the data into a training dataset and test dataset so that the classifier can be evaluated on data it had not seen previously.

We'll divide the data into two portions: 75 percent for training and 25 percent for testing.

```
> # Splitting dataset into train and test datasets
> data_train <- Data[1:1353656, ]
> data_test<-Data[1353657: 1804874,]
```

Then the document-term matrix:

```
> dtm_train <- sms_dtm[1:1353656, ]
> dtm_test  <- sms_dtm[1353657: 1804874, ]
```

Finally, we split the corpus:

```
> corpus_train <- corpus_clean[1:1353656]
> corpus_test  <- corpus_clean[1353657: 1804874]
```

To confirm that the subsets are representative of the complete set of the comment texts, let's compare the proportion of spam in the training and test data frames:

```
> prop.table(table(data_train$target))

 Non-Toxic      Toxic
0.94266416 0.05733584
> prop.table(table(data_test$target))

 Non-Toxic      Toxic
0.93611735 0.06388265
```

Both the training data and test data contain about 7 percent toxic comments. This shows that the toxic messages were divided evenly between the two datasets.

Visualizing text data – word clouds

A word cloud is a way to visually depict the frequency at which words appear in text data. The cloud is made up of words scattered somewhat randomly around the figure.

Words appearing more often in the text are shown in a larger font, while less common terms are shown in smaller fonts.

The “wordcloud” package provides a simple R function to create this type of diagram.

Comparing the word clouds for rejected and approved messages will help us gauge whether our Naive Bayes spam filter is likely to be successful.

```
> library(wordcloud)
> wordcloud(corpus_train, min.freq = 130, random.order = FALSE)
```

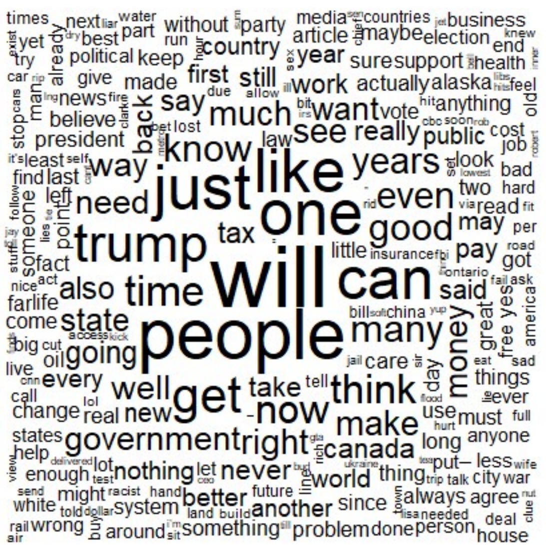
Since we specified random.order = FALSE, the cloud will be arranged in non-random order, with the higher-frequency words placed closer to the center.

The “min.freq” parameter specifies the number of times a word must appear in the corpus before it will be displayed in the cloud.

A general rule is to begin by setting min.freq to a number roughly 10 percent of the number of documents in the corpus; in this case 10 percent is about 130

Therefore, words in the cloud must appear in at least 130 SMS messages.

The resulting word cloud is as follows:



Another important visualization involves comparing the word clouds for comments rejected and approved.

We use R's subset() function to take a subset of the data_train data by comment type. First, we'll create a subset where rating is equal to “rejected”:

```
> rejected <- subset(data_train, rating== "rejected")
```

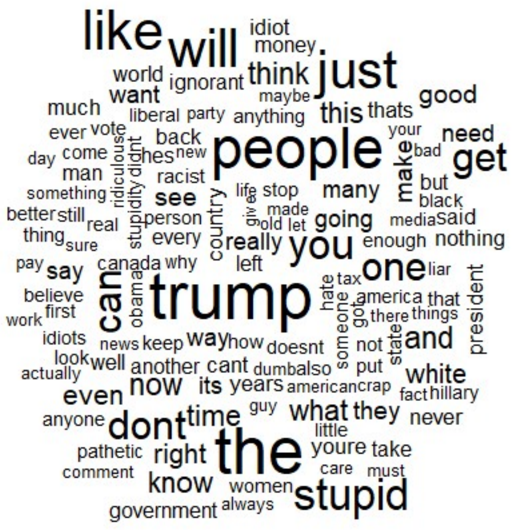
We do the same thing for the “approved” subset:

```
> approved <- subset(data_train, rating== "approved")
```

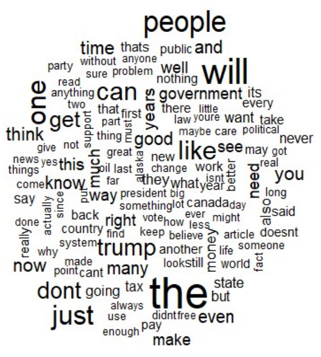
We now have two data frames, “rejected” and “approved”, each with a text feature containing the raw text strings for comment text messages.

We create a word cloud as before. We use the scale parameter to adjust the maximum and minimum font size for words in the cloud.

```
> wordcloud(rejected$comment_text, max.words = 130, scale = c(3, 0.5))
```



```
> wordcloud(approved$comment_text, max.words = 130, scale = c(3, 0.5))
```



Creating indicator features for frequent words

The final step in the data preparation process is to transform the sparse matrix into a data structure that can be used to train a Naive Bayes classifier.

We need to reduce the features that may not be of much help in our classifier. To reduce the number of features, we will eliminate any words that appear in less than five comment texts or less than about 0.1 percent of records in the training data.

Finding frequent words requires use of the findFreqTerms() function in the tm package. This function takes a document term matrix and returns a character vector containing the words appearing at least a specified number of times.

```
> #character vector of the words appearing at least 5 times in the dtm_train
matrix
> dict<-findFreqTerms(dtm_train,5)
> dict
```

We limit our training and test matrixes to only the words in the preceding dictionary.

```
> #save this list of frequent terms for use later
> train<-DocumentTermMatrix(corpus_train,list(dictionary=dict))
> test<-DocumentTermMatrix(corpus_test,list(dictionary=dict))
```

The Naive Bayes classifier is typically trained on data with categorical features. This poses a problem since the cells in the sparse matrix indicate a count of the times a word appears in a message. We change this to a factor variable that simply indicates yes or no depending on whether the word appears at all. The convert_counts() function to convert counts to factors:

```
> convert_counts <- function(x) {
+ x <- ifelse(x > 0, 1, 0)
+ x <- factor(x, levels = c(0, 1), labels = c("No", "Yes"))
+ return(x)
+ }
```

The statement ifelse(x > 0, 1, 0) will transform the values in x so that if the value is greater than 0, then it will be replaced with 1, otherwise it will remain at 0.

The factor command simply transforms the 1 and 0 values to a factor with labels No and Yes. Finally, the newly-transformed vector x is returned.

Now we use the apply() function to apply convert_counts to each of the columns in our sparse matrix. The apply() function allows a function to be used on each of the rows or columns in a matrix. . It uses a MARGIN parameter to specify either rows or columns. MARGIN = 2 is used for the columns. MARGIN = 1 is used for rows.

```
> train <- apply(train, MARGIN = 2, convert_counts)
> test  <- apply(test, MARGIN = 2, convert_counts)
```

Training a model on the data

Now that we have transformed the comment texts into a format that can be represented by a statistical model, we apply the naive Bayes algorithm. The algorithm will use the presence or absence of words to estimate the probability that a given comment is toxic.

```
>library(e1071)
```

```
>classifier <- naiveBayes(train, train$target)
```

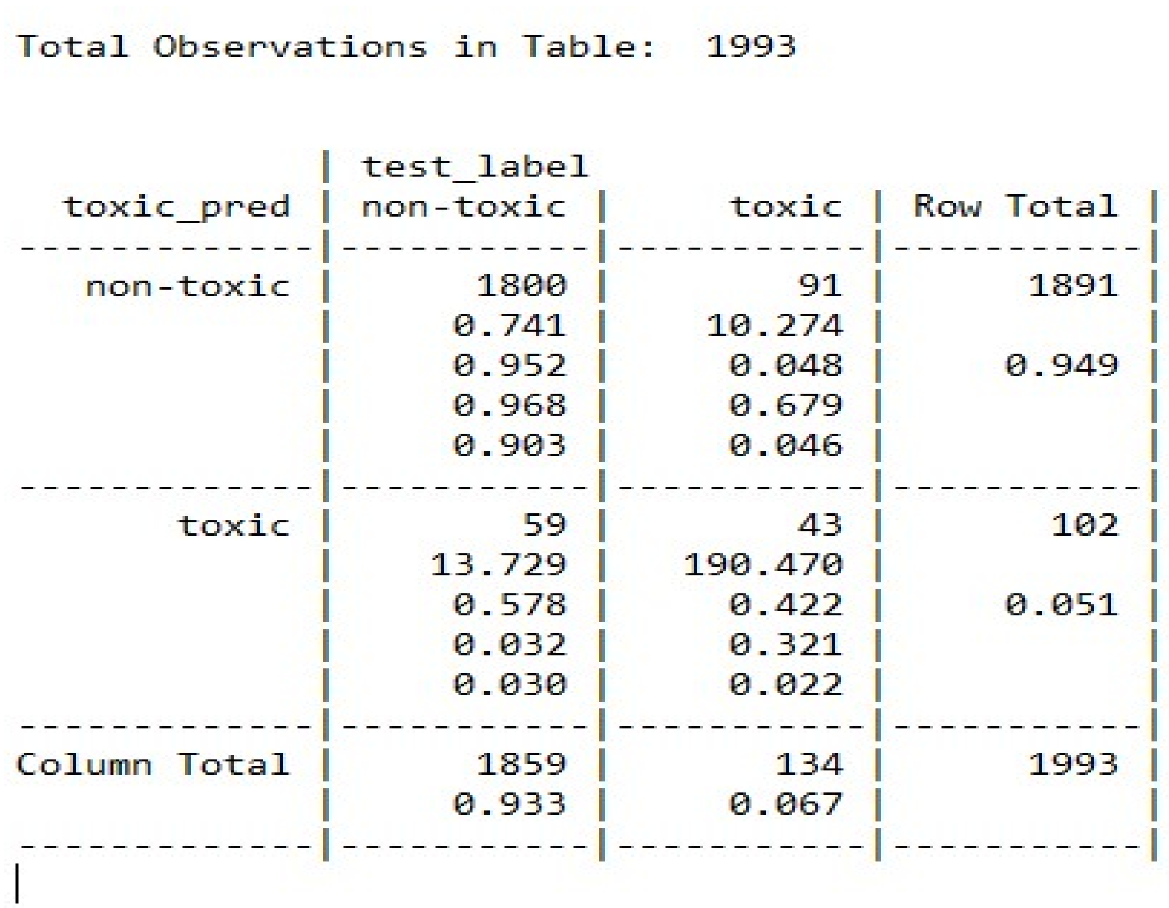
We then use the predict() function to make the predictions.

```
test_pred <- predict(classifier,test)
```

We use the Crosstable() function to print out the confusion matrix.

```
>library(gmodels)
```

```
>CrossTable(test_pred, test$target, prop.chisq = FALSE, prop.t = FALSE)
```



```
Total Observations in Table:  1993


             | test_label
  toxic_pred | non-toxic |     toxic | Row Total |
-------------|-----------|-----------|-----------|
   non-toxic |      1800 |        91 |      1891 |
             |     0.741 |    10.274 |           |
             |     0.952 |     0.048 |     0.949 |
             |     0.968 |     0.679 |           |
             |     0.903 |     0.046 |           |
-------------|-----------|-----------|-----------|
       toxic |        59 |        43 |       102 |
             |    13.729 |   190.470 |           |
             |     0.578 |     0.422 |     0.051 |
             |     0.032 |     0.321 |           |
             |     0.030 |     0.022 |           |
-------------|-----------|-----------|-----------|
Column Total |      1859 |       134 |      1993 |
             |     0.933 |     0.067 |           |
-------------|-----------|-----------|-----------|
|
```

Looking at the table, we can see that 59 of 1859 non-toxic comments texts (3 percent) were incorrectly classified as toxic, while 91 of 134 toxic texts (67.9 percent) were incorrectly classified as non-toxic. This is shown in the summary statistics below:

```
> results <- confusionMatrix(test_pred,test_label)
```

```
> print(results)
Confusion Matrix and Statistics

           Reference
Prediction  non-toxic toxic
  non-toxic      1800    91
  toxic            59    43

               Accuracy : 0.9247
                 95% CI : (0.4871, 0.5201)
    No Information Rate : 0.9328
    P-Value [Acc > NIR] : 0.55964

                  Kappa : 0.3252

 Mcnemar's Test P-Value : 0.01137

            Sensitivity : 0.9683
            Specificity : 0.3209
         Pos Pred Value : 0.9519
         Neg Pred Value : 0.4216
             Prevalence : 0.9328
         Detection Rate : 0.9032
   Detection Prevalence : 0.5278
      Balanced Accuracy : 0.5033

       'Positive' Class : non-toxic
```

Conclusion

In conclusion, as we observe, our model returns an accuracy of 92.5%. However, it's much more effective in classification of the non-toxic comments compared to the toxic. The Kappa statistic value of 0.3252 attests to this. It adjusts accuracy by accounting for the possibility of a correct prediction by chance alone.

It's much useful for datasets with a severe class imbalance, like in our case, because a classifier can obtain high accuracy simply by always guessing the most frequent class. Kappa statistic values range from 0 to a maximum of 1, which indicates perfect agreement between the model's predictions and the true values. They are interpreted on the following scale:

- Poor agreement = less than 0.20
- Fair agreement = 0.20 to 0.40

• Moderate agreement = 0.40 to 0.60

• Good agreement = 0.60 to 0.80

• Very good agreement = 0.80 to 1.00

Our Kappa value of 0.3252, reveals a moderate agreement between the model's predictions and the true values.

Going by this, we need to tune our model for better results

References

1. Brett, L. (2015). *Machine Learning with R*. Birmingham, UK. Packt Publishing Ltd.
2. Peter, B. (2017). Practical Statistics for Data Scientists. Boston, MA. O'Reilly Media, Inc.