PREDICTIVE ANALYTICS

ALY6020, SPRING 2020

MODULE 2 PROJECT ASSIGNMENT

WEEK 2: CLASSIFICATION USING NAÏVE BAYES

SUBMITTED BY: SHIVANI ADSAR

NUID: 001399374

SUBMITTED TO: PROF. NA YU

DATE: 04/18/2020

Introduction

The assignment aims at providing practical experience of working on datasets to perform Naïve Bayes classification algorithm. The machine learning algorithm has been implemented on "Spam" and "IMDB" datasets for classification. The Naïve Bayes is an effective algorithm that uses probabilistic approaches to classify data and perform predictions. [1]

Analysis

Spam Dataset

- The Spam Dataset has been taken from the UCI Machine Learning Repository. This dataset has 5574 instances. The data consists of collection of spam and ham text messages of users.
- In order to perform the classification of the text types as Ham and Spam, we have used the Naïve Bayes Classification algorithm.[1]

<u>Importing the Dataset</u>

- The Spam dataset has been imported initially into a dataframe. Using the "str" function, the structure of the dataset has been analysed into "type" and "text".
- The "type" is a character vector and is a categorical variable. Hence the variable has been factorized. [1]

```
> sms_raw$type <- factor(sms_raw$type)
> str(sms_raw$type)
 Factor w/ 2 levels "ham","spam": 1 1 2 1 1 2 1 1 2 2 ...
> table(sms_raw$type)

 ham spam
4827  747
```

Fig.1: Factorization of Type Variable

We can observe that the "type" variable has been factorized into "ham" and "spam". So, there are 4827 ham texts and 747 spam texts.

<u>Analysis on Dataset</u>

- The texts are collected and saved in a Corpus of documents and then the first 3 documents from the corpus are analyzed.

```
> sms_corpus <- Corpus(VectorSource(sms_raw$text))
> print(sms_corpus)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 5574
> inspect(sms_corpus[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] Go until jurong point, crazy.. Available only in bugis n great world
 wat...
[2] Ok lar... Joking wif u oni...

[3] Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005.
stion(std txt rate)T&C's apply 08452810075over18's
```

Fig.2: Corpus of the text messages

We have created a corpus of texts and inspected the first three documents. Corpus is a collection of documents which makes it easier for document retrieval. The corpus contains a collection of 5574 documents.

- We have used the “tm” package which is used for performing text mining. The data is cleaned by using the “tm_map” function which transforms the tm_corpus function.

```
library(tm)
corpus_clean <- tm_map(sms_corpus, tolower)
corpus_clean <- tm_map(corpus_clean, removeNumbers)
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
sms_dtm <- DocumentTermMatrix(corpus_clean)
```

Fig.3: Data Cleaning

We can see that the data has been cleaned to remove some stop words such as “and”, “to”, “an”etc., the data has been converted to lower case, numbers and punctuations have been removed.

- Using the process of “tokenization”, the messages have been split into individual tokens or words. We have used the DocumentTermMatrix() function which takes the cleaned corpus as the input parameter and creates a matrix. This matrix shows the frequency of words occurring in every document.

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

The DocumentTermMatrix() will convert the corpus into tokens and return a sparse matrix, which will be used for performing further analysis.

Splitting into Training and Testing Dataset

- The data is split into 75 % training and 25% testing datasets, where we train and build our model on the training dataset and validate using the test dataset.[1]
- First, we will split the raw dataset:

```
> sms_raw_train <- sms_raw[1:4169, ]
> sms_raw_test  <- sms_raw[4170:5559, ]
```

Fig.4: Splitting of raw data

We can see that the first 4169 of the data is used for training and the rest of 1390 data is used for testing.

- We have split the document term matrix

```
sms_dtm_train <- sms_dtm[1:4169, ]
sms_dtm_test  <- sms_dtm[4170:5559, ]
```

Fig.5: Splitting of Document Term Matrix

- Further, We have split the corpus into training and testing datasets

```
sms_corpus_train <- corpus_clean[1:4169]
sms_corpus_test  <- corpus_clean[4170:5559]
```

Fig.6: Splitting of Corpus

- In order to view the proportion of ham and spam data in the training and testing data, we have used the prop.table() function.

```
> prop.table(table(sms_raw_train$type))

      ham      spam 
0.8647158 0.1352842 
> prop.table(table(sms_raw_test$type))

      ham      spam 
0.8697842 0.1302158 
```

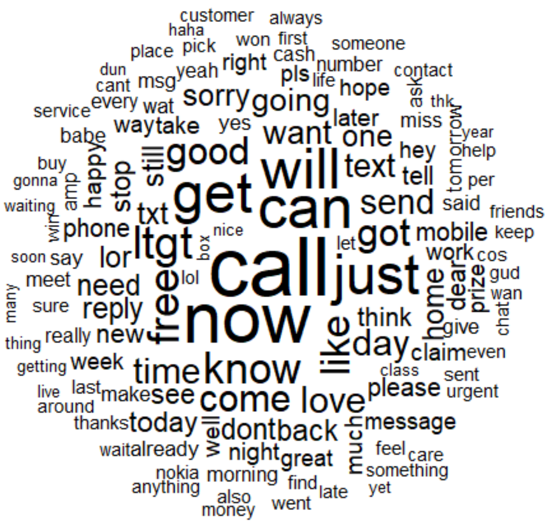
Fig.7: Proportion of ham and spam

We can observe that the data in both the testing and training is almost the same with spam being approximately 13%, hence the data is split equally.

<u>Word Cloud</u>

- We have used word cloud to visualize the frequency of highest occurring words in the document, larger the words, higher is the frequency and smaller words show lesser frequency.[1]
- We have create a word cloud using the word cloud package.

```
wordcloud(sms_corpus_train, min.freq = 40, random.order = FALSE)
```



This word cloud has been created from the sms_corpus_train dataset. Since the minimum frequency has been specified to 40, only the words occurring 40 times in the document will be shown. Also, the function taken data in a non-random order.

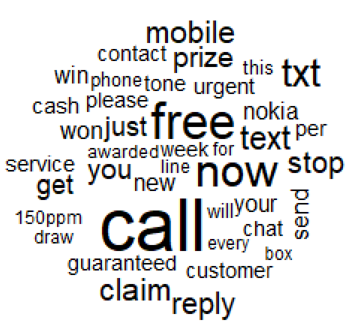
Fig.8: Word Cloud

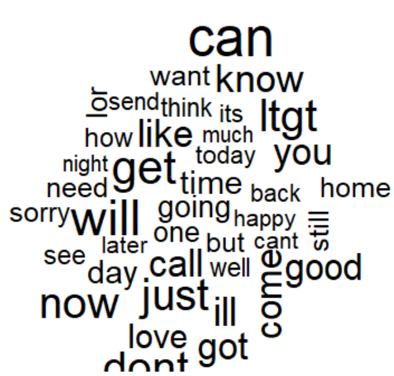
- In order to compare the word clouds of both spam and ham, we have created subsets from the raw dataset.

```
spam <- subset(sms_raw_train, type == "spam")
ham <- subset(sms_raw_train, type == "ham")
wordcloud(spam$text, max.words = 40, scale = c(3, 0.5))
wordcloud(ham$text, max.words = 40, scale = c(3, 0.5))
```

Fig.9: Spam and Ham Datasets

- We have created two different spam and ham datasets using the "text" feature, with maximum words as 40 and scale for adjustment of font.[1]





As observed, the Spam word cloud shows the words like urgent, free, call, now, prize, stop etc. and is easier to detect suspicious texts. However, the Ham word cloud shows words like can, know, time, good etc. which are legitimate texts.

Fig.10: Word Cloud- Spam

Fig.11: Word Cloud- Ham

- We would be using the sparse matrix to train our model for Naïve Bayes classification algorithm. Since, all the features from the sparse matrix will not be considered for classification, will use the findFreqTerms() which will be used for finding the frequent occurring terms in the document. The function takes the matrix as input and then returns a character vector. The results of the function are stored in a dictionary.

```
> findFreqTerms(sms_dtm_train, 5)
  [1] "available"   "bugis"    "cine"     "crazy"    "got"
  [6] "great"       "point"    "wat"      "world"    "lar"
 [11] "wif"         "apply"    "comp"     "cup"      "entry"
 [16] "final"       "free"     "may"      "receive"  "text"
```

Fig.12: Frequency Matrix

```
sms_dict <- c(findFreqTerms(sms_dtm_train, 5))
sms_train <- DocumentTermMatrix(sms_corpus_train,    list(dictionary = sms_dict))
sms_test  <- DocumentTermMatrix(sms_corpus_test,    list(dictionary = sms_dict))
```

<u>Naïve Bayes Classifier</u>

- The Naïve Bayes classifier works well with categorical values. We can see that the values in the matrix are character vector values, hence we need to convert them into factor values.

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

Fig. 13: Conversion of vector into factor values

We can observe that, for factorization, we have converted the values of x, if greater than 0, it will be replaced with 1, else will remain 0. So, the labels of “No” and “Yes” have been given accordingly.

- Further, we have used the apply() function to convert the counts of train and test data for factorization, using Margin = 2 as we are selecting the columns.

```
sms_train <- apply(sms_train, MARGIN = 2, convert_counts)
sms_test  <- apply(sms_test, MARGIN = 2, convert_counts)
```

Fig.14: Converting into factors for Train and Test samples

- For performing the Naïve Bayes classifier, we need to install the “e1071” packages.

```
sms_classifier <- naiveBayes(sms_train, sms_raw_train$type)
sms_test_pred <- predict(sms_classifier, sms_test)
```

Fig.15: Naïve Bayes Classifier

- We have built the Naïve Bayes classifier on the model and used predict() function for predictions. Since we need to evaluate our predictions, we will be comparing with the unseen data which has been stored in sms_test, whereas sms_classifier is our trained classifer.
- Now, we will be comparing the predicted values with actual values by using the CrossTable() function which is available in the gmodels package.

```
CrossTable(sms_test_pred, sms_raw_test$type,    prop.chisq = FALSE, prop.t = FALSE,    dnn = c('predicted',
'actual'))
```

Fig.16: Cross Table function

```
Total Observations in Table:  1390

             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1203 |        28 |      1231 |
             |     0.977 |     0.023 |     0.886 |
             |     0.995 |     0.155 |           |
-------------|-----------|-----------|-----------|
        spam |         6 |       153 |       159 |
             |     0.038 |     0.962 |     0.114 |
             |     0.005 |     0.845 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

Fig. 17: Cross Table Output

We can observe that, out of a total of 1209 ham messages, 6 ham messages were incorrectly classified as spam, this accounts to approx.. 0.4 percent. Whereas, 28 out of a total of 181 spam messages were incorrectly classified as ham, which accounts to 18 percent.

- We can observe that the 6 ham messages were incorrectly classified as spam, this could be a major problem as the classifier will predict ham messages to be spam. Hence we need to improve the performance because sometimes the classifier classifies a particular word to be a spam even if it occurred once.
- We will build a Naïve Bayes model and set a laplace =1.

```
> sms_classifier2 <- naiveBayes(sms_train, sms_raw_train$type,    laplace = 1)
> sms_test_pred2 <- predict(sms_classifier2, sms_test)
> CrossTable(sms_test_pred2, sms_raw_test$type,    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,    dnn
 = c('predicted', 'actual'))
```

Fig.18: Naïve Bayes Classifier with Laplace

```
Total Observations in Table:  1390

             | actual
   predicted |       ham |      spam | Row Total |
-------------|-----------|-----------|-----------|
         ham |      1204 |        30 |      1234 |
             |     0.996 |     0.166 |           |
-------------|-----------|-----------|-----------|
        spam |         5 |       151 |       156 |
             |     0.004 |     0.834 |           |
-------------|-----------|-----------|-----------|
Column Total |      1209 |       181 |      1390 |
             |     0.870 |     0.130 |           |
-------------|-----------|-----------|-----------|
```

We can observe a small improvement of the classification of false positives from 6 to 5.

Fig.19: Cross Table Output of Laplace

- We have observed that the Naïve Bayes classifier is a very efficient technique for classification of ham and spam messages with an accuracy of 98 %.

IMDB Dataset

- The IMDB Dataset has been taken from the Kaggle. This dataset has 4000 instances. The data consists of collection of movie reviews of people.[2]
- In order to perform the classification of the sentiment types as Positive and Negative, we have used the Naïve Bayes Classification algorithm.

<u>Importing the Dataset</u>

- The IMDB dataset has been imported initially into a dataframe. Using the “str” function, the structure of the dataset has been analysed into “Review” and “Sentiment”.
- The “Sentiment” is a character vector and is a categorical variable. Hence the variable has been factorized.

```
> imdb <- read.csv("C:/Users/Shivani Adsar/OneDrive/Desktop/Northeastern University/Predictive Analytics/Modu
le 2/IMDB Dataset.csv")
> str(imdb)
'data.frame':	4000 obs. of  2 variables:
 $ review   : Factor w/ 3998 levels "'Airport 4' is basically a slopped together mess for Universal Studios t
o try and work a new twist - the Concor"| __truncated__,..: 2476 279 1699 532 2526 2556 1666 3566 753 1887
 ...
 $ sentiment: Factor w/ 2 levels "negative","positive": 2 2 2 1 2 2 2 1 1 2 ...
> imdb$sentiment <- factor(imdb$sentiment)
> table(imdb$sentiment)

negative positive 
    2027     1973 
```

Fig.20: Factorization of Sentiment Variable

We can observe that the "sentiment" variable has been factorized into "Positive" and "Negative". So, there are 2027 Negative texts and 1973 Positive texts.

<u>Analysis on Dataset</u>

- The reviews are collected and saved in a Corpus of documents and then the first 3 documents from the corpus are analyzed. [2]

```
> imdb_corpus <- Corpus(VectorSource(imdb$review))
> print(imdb_corpus)
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 4000
> inspect(imdb_corpus[1:3])
<<SimpleCorpus>>
Metadata:  corpus specific: 1, document level (indexed): 0
Content:  documents: 3

[1] One of the other reviewers has mentioned that after watching just 1 Oz episode you'll be hooked. They are
 right, as this is exactly what happened with me.<br /><br />The first thing that struck me about Oz was its
 brutality and unflinching scenes of violence, which set in right from the word GO. Trust me, this is not a s
how for the faint hearted or timid. This show pulls no punches with regards to drugs, sex or violence. Its is
```

Fig.21: Corpus of the movie reviews

We have created a corpus of reviews and inspected the first three documents. Corpus is a collection of documents which makes it easier for document retrieval. The corpus contains a collection of 4000 documents.

- We have used the "tm" package which is used for performing text mining. The data is cleaned by using the "tm_map" function which transforms the tm_corpus function.

```
library(tm)
corpus_clean <- tm_map(imdb_corpus, tolower)
corpus_clean <- tm_map(corpus_clean, removeNumbers)
corpus_clean <- tm_map(corpus_clean, removeWords, stopwords())
corpus_clean <- tm_map(corpus_clean, removePunctuation)
corpus_clean <- tm_map(corpus_clean, stripWhitespace)
```

Fig.22: Data Cleaning

We can see that the data has been cleaned to remove some stop words such as "and", "to", "an"etc., the data has been converted to lower case, numbers and punctuations have been removed.

- Using the process of "tokenization", the messages have been split into individual tokens or words. We have used the DocumentTermMatrix() function which takes the cleaned corpus as the input parameter and creates a matrix. This matrix shows the frequency of words occurring in every document. [2]

```
imdb_dtm <- DocumentTermMatrix(corpus_clean)
```

The DocumentTermMatrix() will convert the corpus into tokens and return a sparse matrix, which will be used for performing further analysis.

<u>Splitting into Training and Testing Dataset</u>

- The data is split into 75 % training and 25% testing datasets, where we train and build our model on the training dataset and validate using the test dataset.

- First, we will split the raw dataset:

```
imdb_raw_train <- imdb[1:3000, ]
imdb_raw_test  <- imdb[3001:4000, ]
```

Fig.23: Splitting of raw data

We can see that the first 3000 of the data is used for training and the rest of 1000 data is used for testing.

- We have split the document term matrix

```
imdb_dtm_train <- imdb_dtm[1:3000, ]
imdb_dtm_test  <- imdb_dtm[3001:4000, ]
```

Fig.24: Splitting of Document Term Matrix

- Further, We have split the corpus into training and testing datasets

```
imdb_corpus_train <- corpus_clean[1:3000]
imdb_corpus_test  <- corpus_clean[3001:4000]
```

Fig.25: Splitting of Corpus

- In order to view the proportion of positive and negative data in the training and testing data, we have used the prop.table() function.[2]

```
> prop.table(table(imdb_raw_train$sentiment))

 negative  positive 
0.4973333 0.5026667 
> prop.table(table(imdb_raw_test$sentiment))

negative positive 
   0.535    0.465 
```

Fig.26: Proportion of Positive and Negative Reviews

We can observe that the data in both the testing and training is almost the same with negative being approximately 50 % and positive being 50%, hence the data is split equally.

Word Cloud

- We have used word cloud to visualize the frequency of highest occurring words in the document, larger the words, higher is the frequency and smaller words show lesser frequency.
- We have create a word cloud using the word cloud package.

```
wordcloud(imdb_corpus_train, min.freq = 50, random.order = FALSE)
```

Fig.27: Word Cloud

This word cloud has been created from the imdb_corpus_train dataset. Since the minimum frequency has been specified to 50, only the words occurring 50 times in the document will be shown. Also, the function taken data in a non-random order.

- In order to compare the word clouds of both the positive and negative reviews, we have created subsets from the raw dataset.

```
positive <- subset(imdb_raw_train, sentiment == "positive")
negative <- subset(imdb_raw_train, sentiment == "negative")
wordcloud(positive$review, max.words = 40, scale = c(3, 0.5))
wordcloud(negative$review, max.words = 40, scale = c(3, 0.5))
```

Fig.28: Positive and Negative Reviews Datasets

- We have created two different Positive and Negative datasets using the “Review” feature, with maximum words as 40 and scale for adjustment of font.[3]

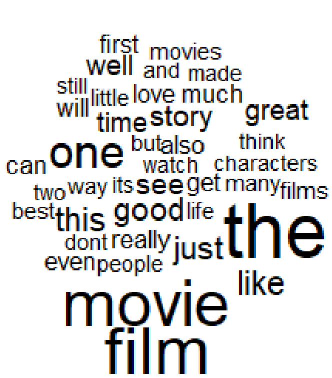


Fig.29: Word Cloud- Positive

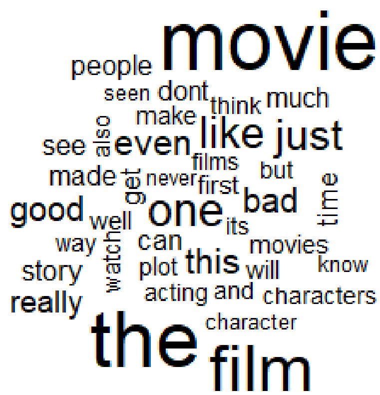


Fig.30: Word Cloud- Negative

As observed, the Positive Reviews word cloud shows the words like great,good, watch, well, movie, best etc. and is easier to understand the comments. However, the Negative Reviews word cloud shows words like bad, dont, never, but etc. which show the negative reviews.

- We would be using the sparse matrix to train our model for Naïve Bayes classification algorithm. Since, all the features from the sparse matrix will not be considered for classification, will use the findFreqTerms() which will be used for finding the frequent occurring terms in the document. The function takes the matrix as input and then returns a character vector. The results of the function are stored in a dictionary.

```
findFreqTerms(imdb_dtm_train, 5)
 [1] "accustomed"     "agenda"       "appeal"        "around"       "audiences"
 [6] "away"           "awaybr"       "become"        "brutality"    "called"
[11] "can"            "cells"        "charm"         "christians"   "city"
[16] "class"          "classic"      "comfortable"   "crooked"      "dare"
imdb_dict <- c(findFreqTerms(imdb_dtm_train, 5))
imdb_train <- DocumentTermMatrix(imdb_corpus_train,   list(dictionary = imdb_dict))
imdb_test  <- DocumentTermMatrix(imdb_corpus_test,   list(dictionary = imdb_dict))
```

Fig.31: Frequency Matrix

<u>Naïve Bayes Classifier</u>

- The Naïve Bayes classifier works well with categorical values. We can see that the values in the matrix are character vector values, hence we need to convert them into factor values. [3]

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

Fig. 32: Conversion of vector into factor values

- Further, we have used the apply() function to convert the counts of train and test data for factorization, using Margin = 2 as we are selecting the columns.

We can observe that, for factorization, we have converted the values of x, if greater than 0, it will be replaced with 1, else will remain 0. So, the labels of “No” and “Yes” have been given accordingly.

```
imdb_train <- apply(imdb_train, MARGIN = 2, convert_counts)
imdb_test  <- apply(imdb_test, MARGIN = 2, convert_counts)
```

Fig.33: Converting into factors for Train and Test samples

- For performing the Naïve Bayes classifier, we need to install the "e1071" packages.

```
imdb_classifier <- naiveBayes(imdb_train, imdb_raw_train$sentiment)
imdb_test_pred <- predict(imdb_classifier, imdb_test)
```

Fig.34: Naïve Bayes Classifier

- We have built the Naïve Bayes classifier on the model and used predict() function for predictions. Since we need to evaluate our predictions, we will be comparing with the unseen data which has been stored in imdb_test, whereas imdb_classifier is our trained classifer.[3]
- Now, we will be comparing the predicted values with actual values by using the CrossTable() function which is available in the gmodels package.

```
> CrossTable(imdb_test_pred, imdb_raw_test$sentiment,    prop.chisq = FALSE, prop.t = FALSE,    dnn = c('pred
icted', 'actual'))
```

Fig.16: Cross Table function

```
Total Observations in Table:  1000

             | actual
   predicted |  negative |  positive | Row Total |
-------------|-----------|-----------|-----------|
    negative |       465 |        84 |       549 |
             |     0.847 |     0.153 |     0.549 |
             |     0.869 |     0.181 |           |
-------------|-----------|-----------|-----------|
    positive |        70 |       381 |       451 |
             |     0.155 |     0.845 |     0.451 |
             |     0.131 |     0.819 |           |
-------------|-----------|-----------|-----------|
Column Total |       535 |       465 |      1000 |
             |     0.535 |     0.465 |           |
-------------|-----------|-----------|-----------|
```

We can observe that, out of a total of 535 negative reviews, 70 negative reviews were incorrectly classified as positive, this accounts to approx.. 13 percent. Whereas, 84 out of a total of 465 positive reviews were incorrectly classified as negative, which accounts to 18 percent.

Fig. 35: Cross Table Output

- We can observe that the 70 negative reviews were incorrectly classified as positive, this could be a major problem as the classifier will predict negative reviews to be positive. Hence we need to improve the performance because sometimes the classifier classifies a particular word to be a spam even if it occurred once.
- We will build a Naïve Bayes model and set a laplace =1.

```
imdb_classifier2 <- naiveBayes(imdb_train, imdb_raw_train$sentiment,    laplace = 1)
imdb_test_pred2 <- predict(imdb_classifier2, imdb_test)
CrossTable(imdb_test_pred2, imdb_raw_test$sentiment,    prop.chisq = FALSE, prop.t = FALSE, prop.r = FALSE,
   dnn = c('predicted', 'actual'))
```

Fig.18: Naïve Bayes Classifier with Laplace

Total Observations in Table: 1000

predicted	actual negative	positive	Row Total
negative	467 0.873	88 0.189	555
positive	68 0.127	377 0.811	445
Column Total	535 0.535	465 0.465	1000

We can observe a small improvement of the classification of false positives from 70 to 68.

Fig.36: Cross Table Output of Laplace

- We have observed that the Naïve Bayes classifier is a very efficient technique for classification of performing sentiment analysis using negative and positive reviews with an accuracy of 87%.

Conclusion

- As per the analysis on the spam dataset and IMDB dataset, we can see that Naïve Bayes is a very effective algorithm, and the accuracy and model performance of the algorithm can be improved by using Laplace.
- Using the Spam Dataset, we have worked on Naïve Bayes classifier to distinguish the spam and ham messages. Also, since the Naïve Bayes algorithm could not classify the ham and ham correctly, we have used laplace to improve the accuracy to 98%.
- Moreover, we have used the IMDB dataset that classified the positive and negative movie reviews of people using Naïve Bayes classifier. In addition, we have used Laplace and could observe a distinct change in accuracy to 87%.
- It can be noted that, Naïve Bayes classifier works well with datasets to classify the categorical data. However, sometimes, the algorithm incorrectly classifies some values and hence it is better to implement the algorithm using Laplace for better performance.[1]

References

1. Lantz, B. (2015). Machine learning with R: learn how to use R to apply powerful machine learning methods and gain an insight into real-world applications. Birmingham: Packt Publ.
2. lakshmi25npathi. (2019, June 19). Sentiment Analysis of IMDB Movie Reviews. Retrieved from https://www.kaggle.com/lakshmi25npathi/sentiment-analysis-of-imdb-movie-reviews
3. Sign In. (n.d.). Retrieved from https://rpubs.com/hoakevinquach/SMS-Spam-or-Ham-Text