measuring effectiveness of knn classification based on preprocessing performed on dataset

Archana balachandran

April 2, 2017

|  |  |  |
| --- | --- | --- |
| **Sl no** | **Topic** | **Page** |
| 1 | Objective and Basic Information | 3 |
| 2 | Experiments | 4-15 |
| 3 | Observations | 16-17 |
| 4 | R code | 18-24 |

**INDEX**

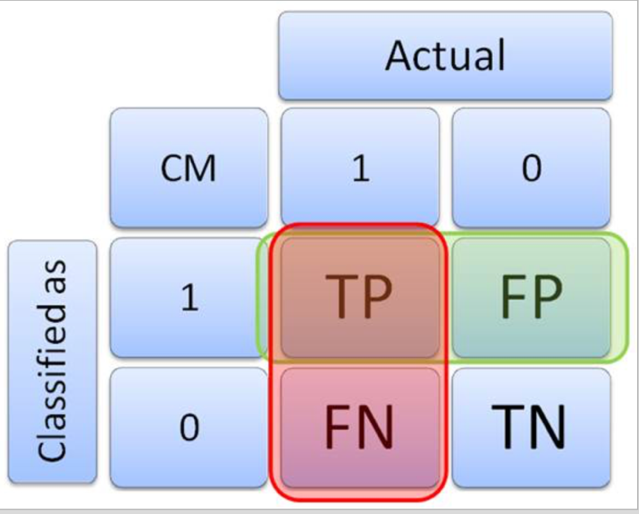
**Objective:** For the newsgroup classification dataset[[1]](#footnote-1), estimate the effectiveness of classification for changes in the preprocessing steps and indicate how the effectiveness of classification changed.

**Information:**

"Rec" considered Positive and "Sci" as Negative

All calculations are tabulated at the end of Part A.

**Confusion Matrix:**



Source: 1 https://onlinecampus.bu.edu/bbcswebdav/pid-4757740-dt-content-rid-16646812\_1/courses/17sprgmetcs688\_o1/module4/allpages.htm

**Experiment 1: No preprocessing**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are skipped, and Document Term Matrix is generated and inspected.
4. Document term matrix is explored
5. Sparse terms are removed
6. Document term matrix is then saved as a simple matrix
7. Splitting Document Term Matrix into training and testing datasets
8. Tags are created
9. KNN classification is performed
10. Confusion matrix is generated using AutoCM for verification purpose
11. Confusion matrix is then manually generated
12. Precision, Recall and f-score values are calculated and the values are stored in exp1result object

**Results:**

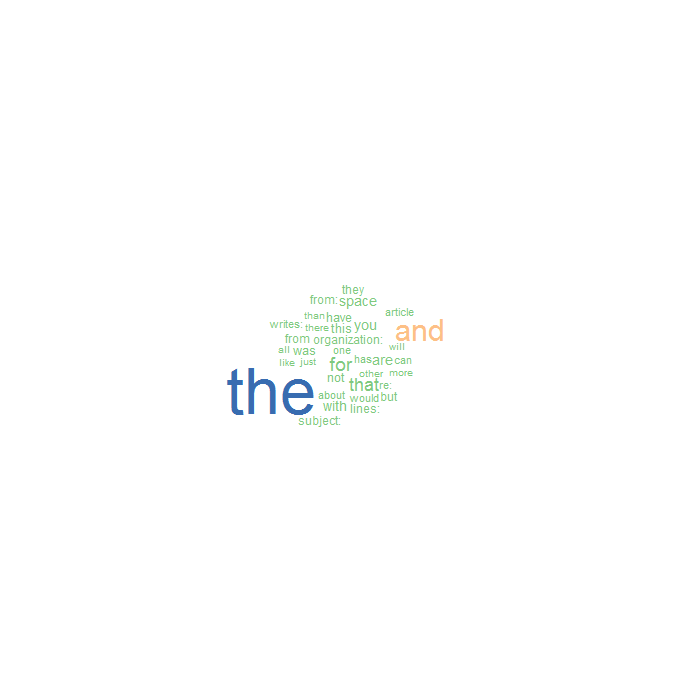


Figure 1 Word Cloud for Document Term Matrix

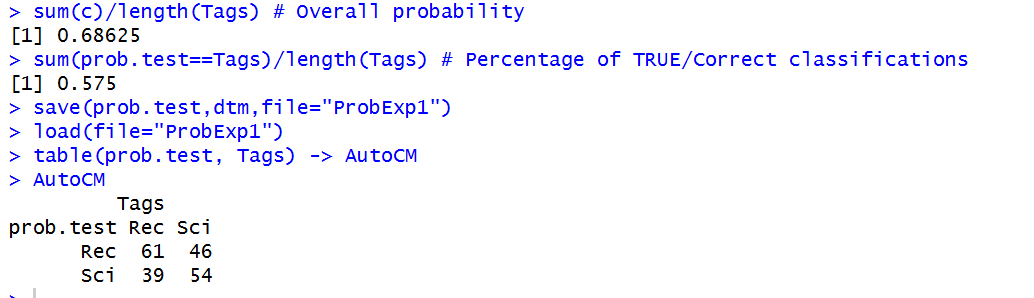


Figure 2 Automatically generated Confusion Matrix

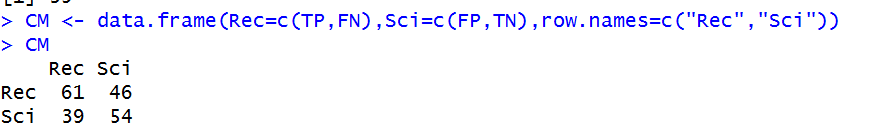


Figure 3 Confusion matrix for data that is nor preprocessed

**Experiment 2: Without Removing sparse terms from DTM**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are skipped, and Document Term Matrix is generated and inspected.
4. Document term matrix is explored
5. Sparse terms are ignored
6. Document term matrix is then saved as a simple matrix
7. Splitting Document Term Matrix into training and testing datasets
8. Tags are created
9. KNN classification is performed
10. Results of classification are analyzed and saved as ProbExp2
11. Confusion matrix is generated using AutoCM for verification purpose
12. Confusion matrix is then manually generated
13. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

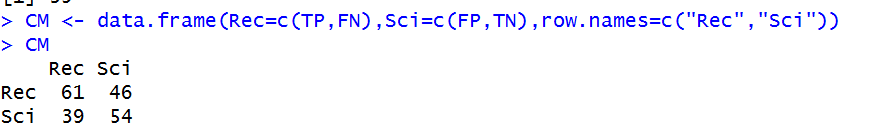


Figure 4 Confusion Matrix for classification Without Removing sparse terms from DTM

**Experiment 3: Preprocessing, without removing stop words**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, skipping stop word removal
4. Document Term Matrix is generated and inspected.
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

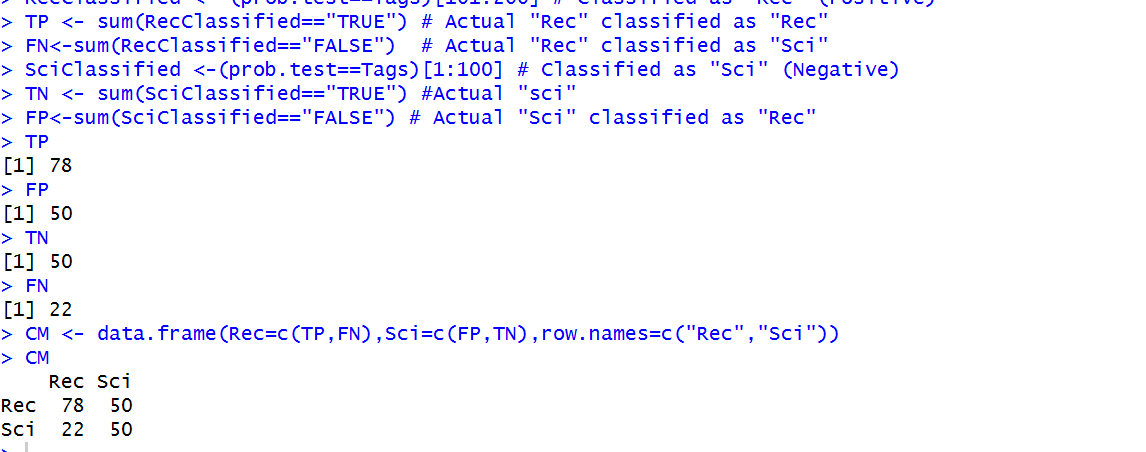
****

Figure Confusion Matrix for classification with preprocessing, without stop word removal

**Experiment 4: Preprocessing, with stop words removal**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, including stop word removal
4. Document Term Matrix is generated and inspected.
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

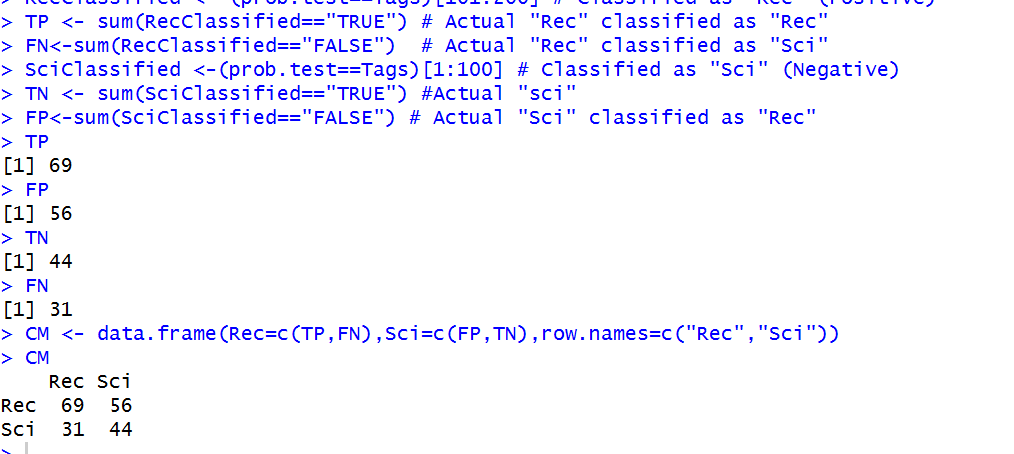


Figure Confusion Matrix for classification with preprocessing, with stop word removal

**Experiment 5: Changing term frequency from 5 to 10**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, including stop word removal
4. Document Term Matrix is generated with term frequency set to 10
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**



Figure Word cloud for experiment 5

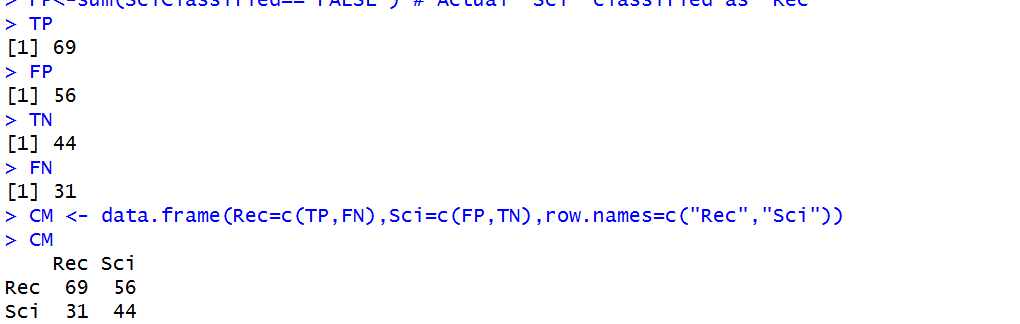


Figure Confusion Matrix for classification with document frequency =10

**Experiment 6: Changing word length from 2 to 4**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, including stop word removal
4. Document Term Matrix is generated with word length set to 4
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

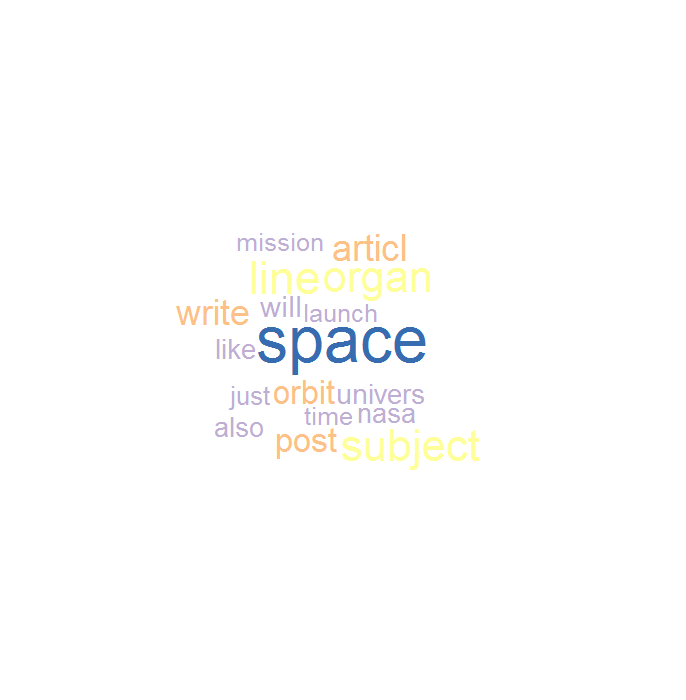


Figure Word Cloud for Experiment 6

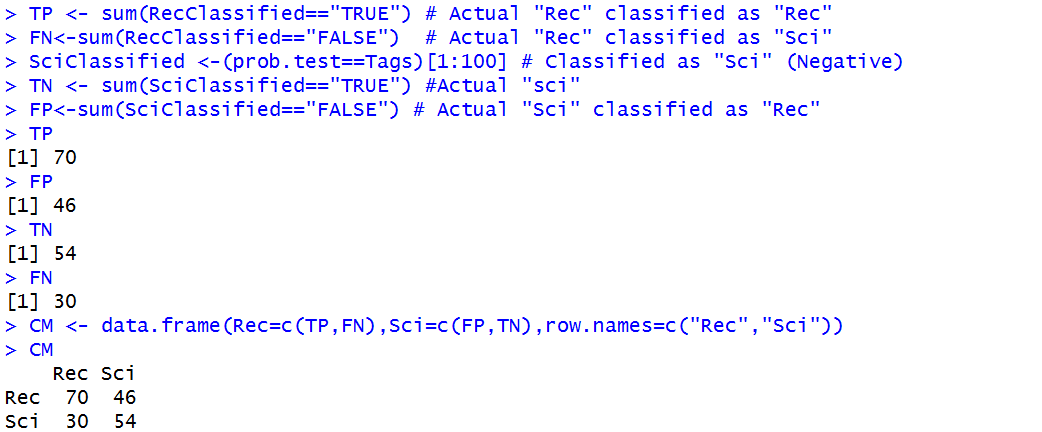


Figure Confusion matrix for classification with wordLength=4

**Experiment 7: Wordlength=4, Document frequency=10**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, including stop word removal
4. Document Term Matrix is generated with word length set to 4 and document frequency as 10
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

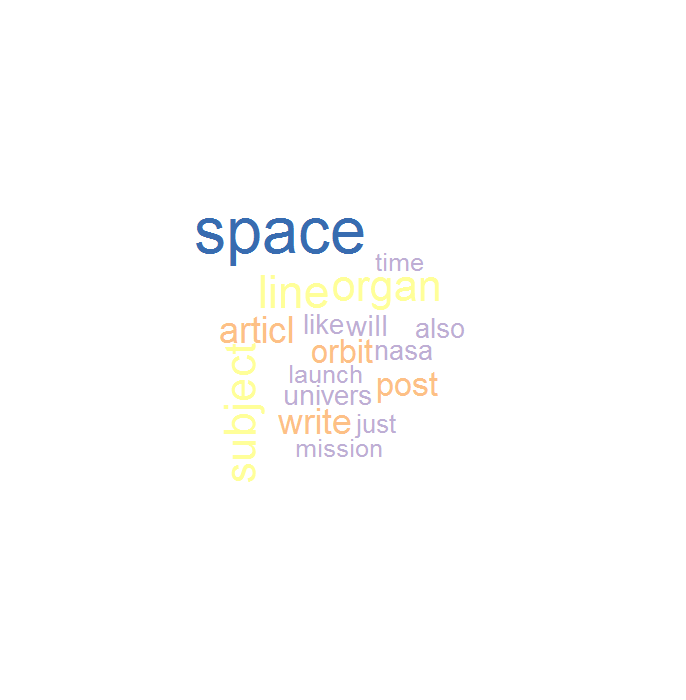


Figure Word cloud for Experiment 7

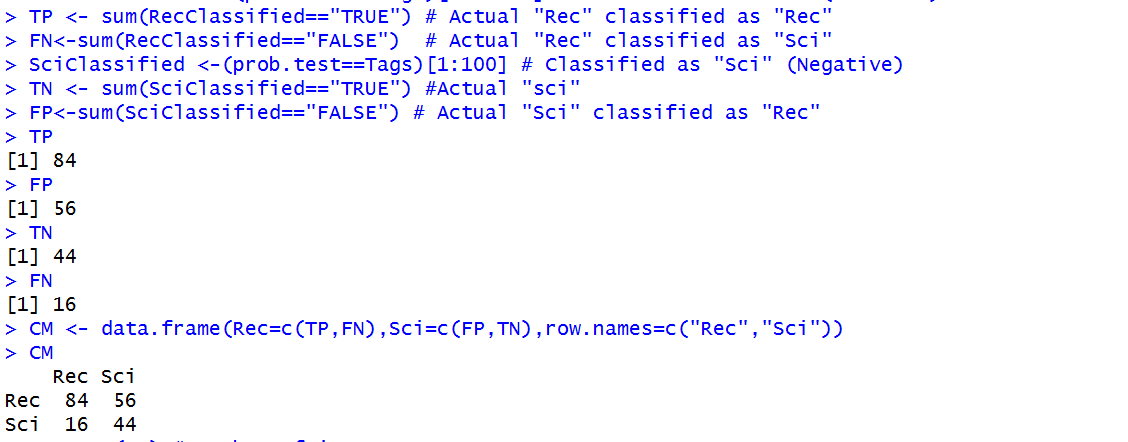


Figure Confusion matrix for wordLength=4, DocFreq=10

**Experiment 8: Wordlength=4, Document frequency=20**

**Steps:**

1. The required libraries are loaded into the workspace
2. The corpus for 2 train document sets and 2 test document sets are created, and then merged
3. Preprocessing steps are performed, including stop word removal
4. Document Term Matrix is generated with word length set to 4 and document frequency as 20
5. Document term matrix is explored
6. Spare terms are removed
7. Document term matrix is then saved as a simple matrix
8. Splitting Document Term Matrix into training and testing datasets
9. Tags are created
10. KNN classification is performed
11. Results of classification are analyzed and saved as ProbExp2
12. Confusion matrix is generated using AutoCM for verification purpose
13. Confusion matrix is then manually generated
14. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

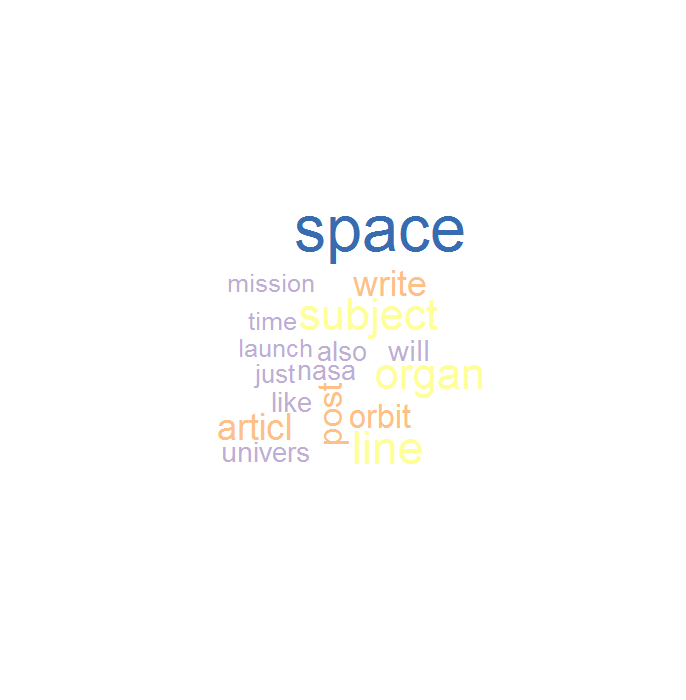


Figure Word cloud for Experiment 8

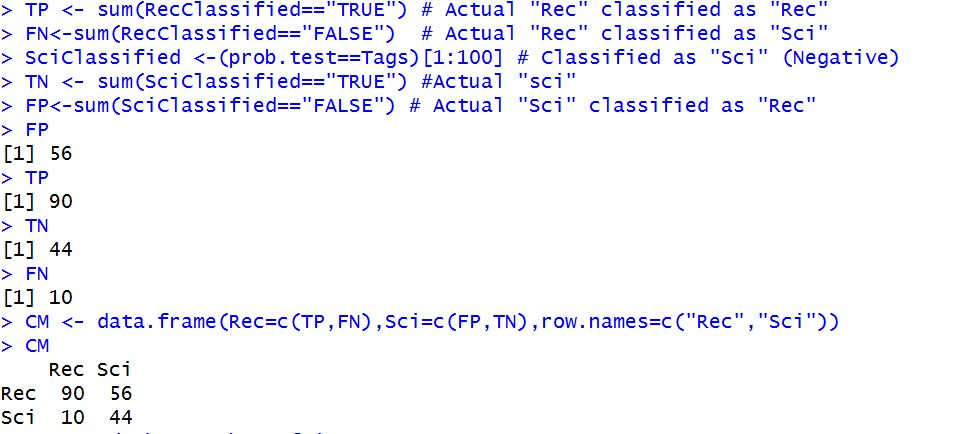


Figure Confusion matrix for Experiment 8

**Experiment 9: Selecting different dataset range (59 to 199)**

1. **Steps:**
2. The required libraries are loaded into the workspace
3. The corpus for 2 train document sets and 2 test document sets are created, and then merged
4. Preprocessing steps are performed, including stop word removal
5. Document Term Matrix is generated with word length set to 4 and document frequency as 20
6. Document term matrix is explored
7. Spare terms are removed
8. Document term matrix is then saved as a simple matrix
9. Splitting Document Term Matrix into training and testing datasets
10. Tags are created
11. KNN classification is performed
12. Results of classification are analyzed and saved as ProbExp2
13. Confusion matrix is generated using AutoCM for verification purpose
14. Confusion matrix is then manually generated
15. Precision, Recall and f-score values are calculated and the values are stored in exp2result object.

**Results:**

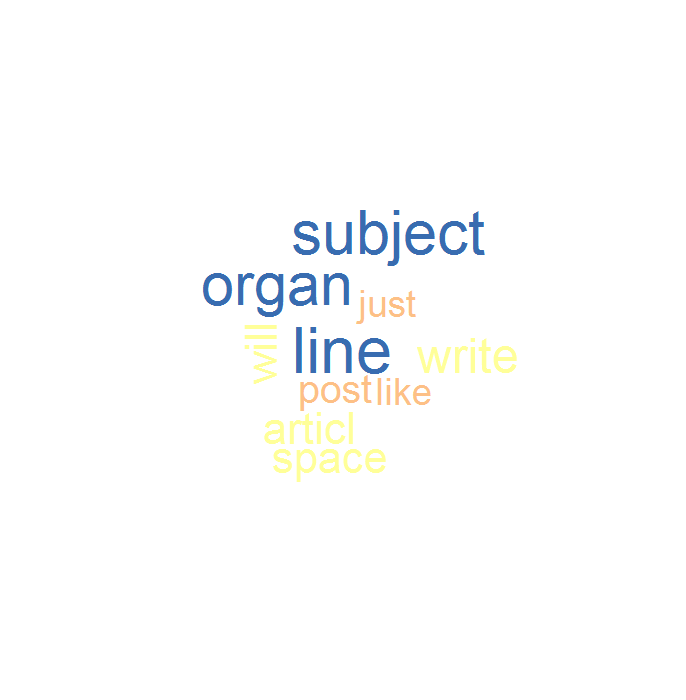


Figure Word Cloud for Experiment 9

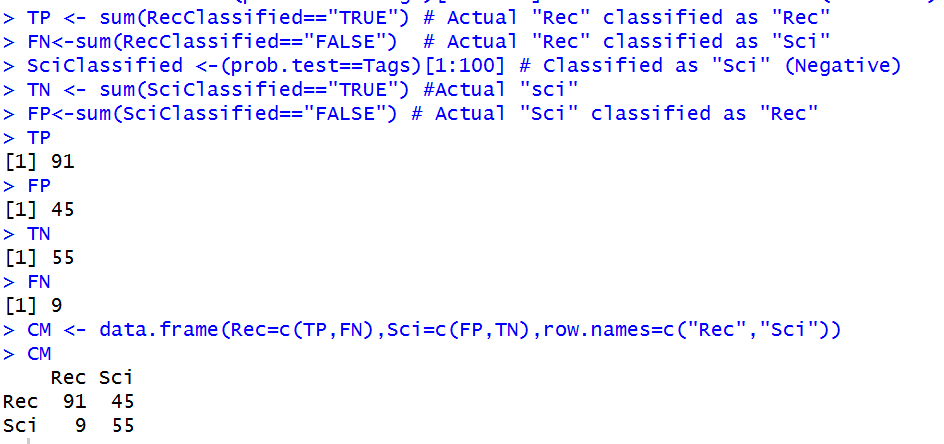
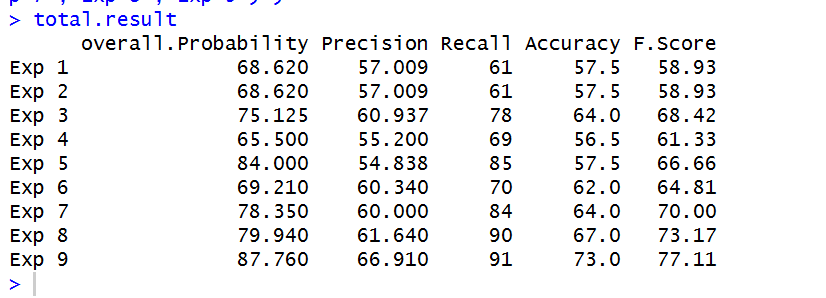


Figure 16 Confusion matrix for Experiment 9

**Observations:**

**Result Dataframe:**



**Tabular form:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Exp** | **Description** | **Overall Probability** | **Precision** | **Recall** | **Accuracy** | **F-Score** |
| 1 | No preprocessing | 68.62 | 57.009 | 61 | 57.5 | 58.93 |
| 2 | No preprocessing, without removing sparse words | 68.62 | 57.009 | 61 | 57.5 | 58.93 |
| 3 | Preprocessing, without stop words removal | 75.125 | 60.937 | 78 | 64 | 68.42 |
| 4 | Preprocessing, with stop words removal | 65.5 | 55.2 | 69 | 56.5 | 61.33 |
| 5 | Term frequency changed from 5 to 10, minWordLength=2 | 84 | 54.838 | 85 | 57.5 | 66.66 |
| 6 | minWordLength=4, DocFreq=5 | 69.21 | 60.34 | 70 | 62 | 64.81 |
| 7 | minWordLength=4, DocFreq=10 | 78.35 | 60 | 84 | 64 | 70 |
| 8 | minWordLength=4, DocFreq=20 | 79.94 | 61.64 | 90 | 67 | 73.17 |
| 9 | Change in dataset: 201:300 range,  DocFreq=20, minWordLength=4 | 87.76 | 66.91 | 91 | 73 | 77.11 |

**Observation:**

* Based on the f-score values, Experiment 9 (f-score =77.11)is the most effective method with highest accuracy. This could also indicate that the newly selected dataset is more relevant for classification.
* Experiment 8 is closely following (with an f-score value on 73.17) with second best accuracy.

It could be related to the word length selection and term frequency values because in both experiments 8 and 9, only those terms which are at least 4 characters in length and occurring in at least 20 documents are selected for creating the document term matrix.

* From Experiments 6,7 and 8, where the minimum word length is set as 4, but the term frequency is varied, we can observe that with increasing term frequency, the f-score values makes a steady increase.
* As shown in the table, Experiments 1 and 2 are least effective and there may be a correlation with the fact that the data is not preprocessed.

**R Code:**

# Author: Archana Balachandran

# --------- Installing and loading required packages-----------

install.packages("tm")

install.packages("SnowballC")

install.packages("wordcloud")

install.packages("class")

library(tm) # for using tm\_map functions

library(SnowballC)

library(wordcloud) # for generating wordcloud

library(class) # for using KNN function

# --------- OBTAINING FILE DIRECTORIES -----------

# Saving the directory source for sci.space.test folder to Temp1 - 394 files

Temp1 <- DirSource("//Mac/Home/Documents/R/win-library/3.3/tm/texts/sci.space.test")

Temp1$filelist[1:100] #verifying that the correct file path is displayed

# Saving the directory source for rec.autos test folder to Temp2 -396 files

Temp2 <- DirSource("//Mac/Home/Documents/R/win-library/3.3/tm/texts/rec.autos.test")

# Saving the directory source for sci.space.train folder to Temp3 - 593 files

Temp3 <- DirSource("//Mac/Home/Documents/R/win-library/3.3/tm/texts/sci.space.train")

# Saving the directory source for rec.autos.train folder to Temp4 - 594 files

Temp4 <- DirSource("//Mac/Home/Documents/R/win-library/3.3/tm/texts/rec.autos.train")

# ------------CORPUS GENERATION -----------------

# Creating the corpus for sci.space.train with 100 elements/files

Doc1.Train <- Corpus(URISource(Temp3$filelist[201:300]),readerControl=list(reader=readPlain))

# Creating the corpus for sci.space.test with 100 elements/files

Doc1.Test <- Corpus(URISource(Temp1$filelist[201:300]),readerControl=list(reader=readPlain))

# Creating the corpus for rec.autos.train with 100 elements/files

Doc2.Train <- Corpus(URISource(Temp4$filelist[201:300]),readerControl=list(reader=readPlain))

# Creating the corpus for rec.autos.test with 100 elements/files

Doc2.Test <- Corpus(URISource(Temp2$filelist[201:300]),readerControl=list(reader=readPlain))

# Merging all of 4 Corpora into 1 Corpus so all pre processing steps can be implemented at once and create one DTM.

# Obtaining merged corpus

Doc.Corpus<-c(Doc1.Train,Doc1.Test,Doc2.Train,Doc2.Test)

# ------------------ PREPROCESSING ----------------

# Objective: To apply transformations across all documents within a corpus, using tm\_map()

# NOTE:  original Doc.Corpus is preserved for backtracking purposes, doc.tranf will undergo transformations

# Two functions used in this section: getTransformations() and content\_transformer()

#    Steps performed:

# 1. converting the text to lowercase

# 2. Removing \t

# 3. Convert to plaintext

# 4. Transform @,-,: to white space

# 5. Stripping whitespace

# 6. Removing stop words

# 7. Removing punctuation - removes , and .

# 8. Removing numbers

# 9. Performing STEMMING

# 1 Transforming to lower case

doc.tranf <-tm\_map(Doc.Corpus,content\_transformer(tolower))

doc.tranf[[1]]$content[1:10]

#Studying a sample document to understand what patterns to eliminate

doc.tranf[[1]]$content[1:10]

# REMOVE <.+?> - Omitted since it gives negative result -displays unnecessary spaces between every letter

# transform.char1<-content\_transformer(function(x,pattern) gsub(pattern," ",x))

# doc.tranf <-tm\_map(doc.tranf, transform.char1,"<|?|>")

# doc.tranf[[1]]$content[1:10]

# 2. REMOVE \t

transform.tab<-content\_transformer(function(x,pattern) gsub(pattern," ",x))

doc.tranf <-tm\_map(doc.tranf, transform.tab,"\t")

doc.tranf[[1]]$content[1:10]

# 3. TO PLAINTEXT

doc.tranf <- tm\_map(doc.tranf, PlainTextDocument)

doc.tranf[[1]]$content[1:10]

# 4. Transforming @,-,: to white space

transform.char2<-content\_transformer(function(x,pattern) gsub(pattern," ",x))

doc.tranf <-tm\_map(doc.tranf, transform.char2,"@|:|-")

doc.tranf[[1]]$content[1:10]

# 5. Stripping whitespace

doc.tranf <-tm\_map(doc.tranf, stripWhitespace)

doc.tranf[[1]]$content[1:10]

# 6. Removing stop words

doc.tranf <- tm\_map(doc.tranf,removeWords,stopwords("english"))

doc.tranf[[1]]$content[1:10]

# 7. Removing punctuation - removes , and .

doc.tranf <- tm\_map(doc.tranf,removePunctuation)

doc.tranf[[1]]$content[1:10]

# 8. Removing numbers

doc.tranf <- tm\_map(doc.tranf,removeNumbers)

doc.tranf[[1]]$content[1:10]

# 9. Performing STEMMING

doc.tranf <-tm\_map(doc.tranf,stemDocument)

doc.tranf[[1]]$content[1:10]

#------------CREATING DOCUMENT TERM MATRIX--------------

# A document-term matrix is a matrix with documents as the rows,

# terms as the columns, and a count of the frequency of words as the cells.

# In the tm package, DocumentTermMatrix() is used to create this matrix.

# To inspect the document-term matrix, inspect() is used.

# SOURCE: onlinecampus.bu.edu/CS688/module3

?DocumentTermMatrix

# Only for experiment 1 and 2

# dtm = DocumentTermMatrix(Doc.Corpus,  control = list(minWordLength = 2,minDocFreq = 5))

# Only for Experiment 5

# dtm = DocumentTermMatrix(doc.tranf,  control=list(wordLengths=c(2, 15), bounds = list(global = c(10,Inf))))

# Only for Experiment 6

# dtm = DocumentTermMatrix(doc.tranf,  control=list(wordLengths=c(4, 15), bounds = list(global = c(5,Inf))))

# Only for Experiment 7

# dtm = DocumentTermMatrix(doc.tranf,  control=list(wordLengths=c(4, 15), bounds = list(global = c(10,Inf))))

# Only for Experiment 8 and 9

# dtm = DocumentTermMatrix(doc.tranf,  control=list(wordLengths=c(4, 15), bounds = list(global = c(20,Inf))))

dtm = DocumentTermMatrix(doc.tranf,

control = list(minWordLength = 2,

minDocFreq = 5))

# exploring the documentterm matrix

inspect(dtm)

freq<-colSums(as.matrix(dtm)) # Term frequencies

ord<-order(freq) # Ordering frequencies

freq[tail(ord)] # most frequent terms

findFreqTerms(dtm,lowfreq = 400) # finding frequent terms having at least 200  occurrences

set.seed(123)

wordcloud(names(freq),freq,min.freq = 200, colors = brewer.pal(5,"Accent"))

# Removing Sparse terms from DocumentTermMatrix with sparse=0.60

?removeSparseTerms

removeSparseTerms(dtm, 0.60)

#saving dtm as simple matrix

matdtm <- as.matrix(dtm)

write.csv(matdtm,file="dtm.csv")

ncol(matdtm)

#------------- GENERATING TRAINING AND TESTING DATASETS FROM  Document Term Matrix--------------

# Splitting DocumentTermMatrix into train dataset and verifying

matdtm[c(1:100),]

train.dataset<-rbind(matdtm[c(1:100),],matdtm[c(201:300),])

#View(train.dataset[c(1:200),c(1:4)])

# Splitting DocumentTermMatrix into test dataset and verifying

test.dataset <-rbind(matdtm[c(101:200),],matdtm[c(301:400),])

#View(test.dataset[c(1:200),c(1:4)])

#    ------- CLASSIFICATION PROCESS ----------

# CREATING tags using rep()

nrow(test.dataset)

nrow(train.dataset)

ncol(test.dataset)

ncol(train.dataset)

Tags <- factor(c(rep("Sci",100),rep("Rec",100)))

# Classifying text using KNN from package class

prob.test <- knn(train.dataset, test.dataset, Tags, k = 2, prob=TRUE)

# Analyzing the output of knn()

a<-1:length(prob.test)

a

b<-levels(prob.test)[prob.test]

b

c<-attributes(prob.test)$prob

c

result<-data.frame(Doc=a, Predict=b, Prob=c, Correct=(prob.test==Tags))

result

overall.prob<-(sum(c)/length(Tags))\*100 # Overall probability

overall.prob

sum(prob.test==Tags)/length(Tags) # Percentage of TRUE/Correct classifications, i.e., the accuracy of classifiication

# Saving the required objects into a file for ease of experimenting

save(prob.test,dtm,file="ProbExp1")

load(file="ProbExp1")

table(prob.test, Tags) -> AutoCM # Automatically Generating CM, only for verification purpose

# --------EVALUATING EFFECTIVENESS OF CLASSIFICATION-------------

# Creating TP, FP, FN, TN

# "Rec" considered Positive and "Sci" as Negative

RecClassified <- (prob.test==Tags)[101:200] # Classified as "Rec" (Positive)

TP <- sum(RecClassified=="TRUE") # Actual "Rec" classified as "Rec"

FN<-sum(RecClassified=="FALSE")  # Actual "Rec" classified as "Sci"

SciClassified <-(prob.test==Tags)[1:100] # Classified as "Sci" (Negative)

TN <- sum(SciClassified=="TRUE") #Actual "sci"

FP<-sum(SciClassified=="FALSE") # Actual "Sci" classified as "Rec"

TP

FP

TN

FN

# Creating the Confusion Matrix

CM <- data.frame(Rec=c(TP,FN),Sci=c(FP,TN),row.names=c("Rec","Sci"))

CM

# Computing the evaluation metrics.

n = sum(CM) # number of instances

diag = TP+TN # number of correctly classified instances per class

# Calculating Accuracy - the fraction of instances that are correctly classified.

accuracy = (sum(diag) / n )\*100

accuracy

# Calculating Precision and Recall

# Precision – The fraction of the returned results that are relevant to the information need

# Recall – The fraction of the relevant documents in the collection that were returned by the system

precision<-(TP/(TP+FP))\*100

precision

recall<-(TP/(TP+FN))\*100

recall

#Calculating f score

fscore<- (2\*precision\*recall)/(precision+recall)

fscore

# Results

overall.Probability<-c(68.62,68.62,75.125,65.5,84.00,69.21,78.35,79.94,87.76)

Precision<-c(57.009,57.009,60.937,55.20,54.838,60.34,60.00,61.64,66.91)

Recall<-c(61.00,61.00,78.00,69.00,85.00,70.00,84.00,90.00,91.00)

Accuracy<-c(57.50,57.50,64.00,56.50,57.50,62.00,64.00,67.00,73.00)

F.Score <- c(58.93,58.93,68.42,61.33,66.66,64.81,70.00,73.17,77.11)

# Creating a data frame with all the results:

total.result<-data.frame(overall.Probability, Precision,Recall,Accuracy,F.Score, row.names =c("Exp 1","Exp 2","Exp 3","Exp 4","Exp 5","Exp 6","Exp 7","Exp 8","Exp 9") )

1. Source: www.csail.mit.edu/~jrennie/20Newsgroups/ [↑](#footnote-ref-1)