# Homework 2 Question 2 Naive Bayes

*Bejan Sadeghian*

*August 18, 2015*

## Task

Use a model to determine the author of a Reuters' article based on a corpus of 2500 documents from 50 authors.

## My Method

For the first model, I decided to use Naive Bayes. In order to import the data I followed along closely to the provided script to collect the authors and their file paths. I did make adjustments to fit my model's needs as necessary. Like the provided code, I replaced whitespaces, removed punctuations and numbers, and lower cased everything similar to what was done in class. I tried using stemming as well but it resulted in a lower accuracy when predicting.

My optimization metric was to improve out of bag prediction accuracy as best as possible so that was the basis for each decision I made. The decisions I made were to use TFIDF as opposed to raw counts, not using normalization on the TFIDF values, and remove sparse terms used less than 1% of the time (the more data I could use the better prediction I typically got, but using 100% of the data made the script very slow).

To deal with words that showed up in the test set that were not in the training set what I did was intersect the two matrices and added whatever words that were missing from the intersection, that was in the train set. I counted those as zeros since they did not appear once in the test set. What this essentially did was drop any words that were not in the training set so I could use my previous data set to predict on the test set. The limitation to this is that if the author started using unique words in the test set that were not in the training set I would not catch that. From my reading online this is a common method but Im sure there are better methods.

Finally I cycled through each smooth factor for each author and multiplied it's log to the vector of each test observation. Whichever author's score produced the largest weight was the one I had my code flag as the author which saved into a dataframe for summary.

The correct prediction percentage I was able to achieve was around 62% which I was proud of. Initially I was getting somewhere on the order of 20-30% but what had gotten me was, even though Naive Bayes does not use order for its calculation of the score factor, because my test set's columns were not ordered similar to the train set it was giving me a poor prediction. (Since my score factor for each author is a vector where it set the values according to the order of the train set's columns). Basically a mismatch when multiplying the weight for each test observation to the score factor vector parameters. When I ordered them it jumped up to around 50%, the optimizations I mentioned earlier got me to ~62%.

At the very bottom you can see that there are six authors that are commonly mistaken for other authors… David Lawder, Jane Macarthy, Scott Hillis, Darren Schuettler, Peter Humphrey, and Todd Nissen.

# Notes

Please be aware that my second method for Question 2 is a seperate R and markdown file. Of the two methods I prefer Naive Bayes because the accuracy for prediction is 10% higher (60% for Naive Bayes vs 50% for Random Forest).

```
library(tm)
```

```
## Loading required package: NLP
```

```
library(stringr)
library(plyr)
library(SnowballC)


## Rolling two directories together into a single corpus
author_dirs = Sys.glob('../STA380 Homework2/ReutersC50/C50train/*')
file_list = NULL
labels = NULL
authors_list = NULL
files_to_add = NULL

readerPlain = function(fname){
  readPlain(elem=list(content=readLines(fname)),
            id=fname, language='en') }

for(i in author_dirs){
  author = i
  author_name = substring(author, first=41)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  labels = append(labels, rep(author_name, length(files_to_add)))
}

for(j in author_dirs){
  authors_list = append(authors_list, substring(j, first=41))
}

#Pieces of this code was provided by Dr. James Scott's Naive Bayes R file.

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
my_corpus = Corpus(VectorSource(all_docs))
```

```
names(my_corpus) = labels


# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower))
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers))
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation))
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace))
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords('SMART'))
my_corpus = tm_map(my_corpus, stemDocument)




#Creating a DTM and a list of all terms used
DTM = DocumentTermMatrix(my_corpus, control= list(weighting = function(x) weightTfIdf(x,
normalize = FALSE)))
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 21542)>>
## Non-/sparse entries: 388227/53466773
## Sparsity           : 99%
## Maximal term length: 36
## Weighting          : term frequency - inverse document frequency (tf-idf)
```

```
DTM = removeSparseTerms(DTM, 0.99) #Removing anything uses less than 1% of the time
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 2431)>>
## Non-/sparse entries: 315775/5761725
## Sparsity           : 95%
## Maximal term length: 20
## Weighting          : term frequency - inverse document frequency (tf-idf)
```

```
ListofTrainTerms = DTM$dimnames
ListofTrainTerms = ListofTrainTerms$Terms

X = as.matrix(DTM) #Full train corpus

smoothCount = 1/50
count = 1

for(i in authors_list){
  #assign(paste0(substring(i, first = 29),'_train'), X[count:(count+49),])
  smoothFactor = colSums(X[count:(count+49),] + smoothCount)
  smoothFactor = smoothFactor/sum(smoothFactor)
  assign(paste0(i,'_SF'),smoothFactor)
  count = count + 50
```

```
}


#Compare against test set

author_dirs = Sys.glob('../STA380 Homework2/ReutersC50/C50test/*')
file_list = NULL
labels = NULL
author_list = NULL
files_to_add = NULL

for(i in author_dirs){
  author = i
  author_name = substring(author, first=40)
  files_to_add = Sys.glob(paste0(author, '/*.txt'))
  file_list = append(file_list, files_to_add)
  for(ix in seq(1,50)){
    author_list = append(author_list, author_name)
  }
  labels = append(labels, rep(author_name, length(files_to_add)))
  names_of_files = str_sub(files_to_add, -16)
}

#Pieces of this code was provided by Dr. James Scott's Naive Bayes R file.

all_docs = lapply(file_list, readerPlain)
names(all_docs) = file_list
names(all_docs) = sub('.txt', '', names(all_docs))
my_corpus = Corpus(VectorSource(all_docs))
names(my_corpus) = labels

# Preprocessing
my_corpus = tm_map(my_corpus, content_transformer(tolower))
my_corpus = tm_map(my_corpus, content_transformer(removeNumbers))
my_corpus = tm_map(my_corpus, content_transformer(removePunctuation))
my_corpus = tm_map(my_corpus, content_transformer(stripWhitespace))
my_corpus = tm_map(my_corpus, content_transformer(removeWords), stopwords('SMART'))
my_corpus = tm_map(my_corpus, stemDocument)


#Creating the DTM and a list of all of the terms used
DTM = DocumentTermMatrix(my_corpus, control= list(weighting = function(x) weightTfIdf(x,
normalize = FALSE)))
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 22088)>>
## Non-/sparse entries: 394051/54825949
## Sparsity           : 99%
## Maximal term length: 45
## Weighting          : term frequency - inverse document frequency (tf-idf)
```

```
DTM = removeSparseTerms(DTM, 0.99) #Removing anything uses less than 1% of the time
DTM
```

```
## <<DocumentTermMatrix (documents: 2500, terms: 2455)>>
## Non-/sparse entries: 320256/5817244
## Sparsity           : 95%
## Maximal term length: 15
## Weighting          : term frequency - inverse document frequency (tf-idf)
```

```
ListofTestTerms = DTM$dimnames
ListofTestTerms = ListofTestTerms$Terms

X_test = as.matrix(DTM)

#Combining the terms to make sure the test DTM has only terms that are also in the train
set
Combined = data.frame(X_test[,intersect(colnames(X_test),colnames(X))])
temp.table = read.table(textConnection(''), col.names=colnames(X),colClasses='integer')
X_eval = rbind.fill(Combined, temp.table)
X_eval[is.na(X_eval)] = 0
X_eval = X_eval[,sort(names(X_eval))]
X_eval = as.matrix(X_eval)

#Calculating probabilties and making a prediction

df = data.frame(Document=rep(0,2500), Predict=rep(0,2500), Weight=rep(0,2500), Actual=re
p(0,2500), Match=rep(0,2500))
df[,4] = author_list
count = 1
index = 1
for(doc in file_list){
  current_best = -10000000000
  for(person in authors_list){
    temp_w = sum(X_eval[count,]*log(get(paste0(person,'_SF'))))
    if (temp_w > current_best){
      current_person = person
      current_best = temp_w
    }
  }
  df[index,1] = str_sub(doc,-15)
  df[index,2] = current_person #Best Prediction
  df[index,3] = current_best
  df[index,5] = ifelse(df[index,4] == df[index,2], 1, 0)
  count = count + 1
  index = index + 1
}


#Calculate Accuracy
Accuracy.Score = sum(df[[5]])/length(df[[5]])
print(Accuracy.Score)
```

```
## [1] 0.6228
```

```
#Look at the tabulation and print which authors are very often mixed with another author.
crosstabulate = xtabs(~Actual + Predict, data=df)
cross.df = as.data.frame.matrix(crosstabulate)
rownames = rownames(crosstabulate)
colnames = rownames(crosstabulate)
for(i in seq(1,50)){
  if(colnames[[i]] != rownames[which(cross.df[[i]]==max(cross.df[[i]]))]){
    print(paste0(rownames[which(cross.df[[i]]==max(cross.df[[i]]))],' is commonly mistake
n for ',colnames[[i]]))
  }
}
```

```
## [1] "ToddNissen is commonly mistaken for DavidLawder"
## [1] "DarrenSchuettler is commonly mistaken for HeatherScoffield"
## [1] "ScottHillis is commonly mistaken for JaneMacartney"
## [1] "JaneMacartney is commonly mistaken for ScottHillis"
## [1] "PeterHumphrey is commonly mistaken for TanEeLyn"
## [1] "DavidLawder is commonly mistaken for ToddNissen"
```