CASE STUDY 2B

UN 5550- Introduction to Data Science

Authors : Rachel Connors, Chathura Gunasekara, Shivam Juneja

DECEMBER 9, 2014

PART I: Party Affiliation Classification

Steps:

A label vector was created using a Perl script to read the speech and create the training set of each president and their respective party

The Perl script creates bag of words for each speech and at the same time, eliminates any words that are listed in the provided file stopwords.txt. This prevents them from being counted in the bag of words analysis.

The Bernoulli model was used to predict the respective party affiliation for four presidential speeches. This method examines each bag of words as having binary elements with 1 representing the word is present and 0 meaning it is not present.

The script in Appendix A created two bag of word files, one for democratic speeches and one for republican. The two files were sorted by the word frequency using a bash script and the top 300 words from each file were extracted. Then, only the unique words were selected to create the feature vector. The number of words present creates the vocabulary V.

In Naïve Bayes model, there are various calculations that must be performed prior to analyzing the data. These include the number of documents, the number of documents of each class, and the number of documents in each class that have a particular word repeated for every word in the vocabulary V[1].

Probabilities can be used to help predict the party by examining the number of times a particular word appears. For example, if the word "freedom" appears in 23 documents out of 50 democratic documents the probability can be represented as P(freedom|democratic)=23/50. This would also be calculated for how many times the word "freedom" appears in a republican document.

A Perl script was used to generate the Bernoulli data set, shown in table 1. Using the code in *Bernoulli_model.R*, the predicted results using the Bernoulli data set are shown in table 2.

work	world	year	years	Party
1	1	0	1	Democratic
1	0	1	0	Republican
0	1	0	1	Democratic
1	0	1	0	Republican
0	1	1	1	Democratic

Table 1. Bernoulli data set

Part I Task 3. (b).

```
train<-read.csv("./train/bernouli.csv")
nzro<-nearZeroVar(train[,-360])
train<-train[,-nzro]
test<-read.csv("./test/bernouliTest.csv")
test<-test[,-nzro]

#calculating prior probablilities
prior <-table(train[,341])/length(train[,341])

#calculating p(evidence) for each word in Democratic and republican. Then calculate the l
ikelyhoods p(word|Dems) and p(word|Rep)


#Laplace smoothing has been done by adding 1 when calculating the likely hood to prevent
a likelyhood probabality to becomig zero. For example if a word does not appear atleast o
ne time in Democratic documents. p(word|Democratic)=0.

trainDem<- subset(train,train[,341]=="Democratic")
trainRep<- subset(train,train[,341]=="Republican")
demVec<-c()
repVec<-c()
for(i in 1:340){
  pWordgivenParty<-(sum(as.numeric(trainDem[,i]))+1)/length(trainDem[,341])
  demVec<-c(demVec,pWordgivenParty)
}
for(i in 1:340){
  pWordgivenParty<-(sum(as.numeric(trainRep[,i]))+1)/length(trainRep[,341])
  repVec<-c(repVec,pWordgivenParty)
```

Computing the posterior probabilities for test cases

1. Barak Obama 2014

```
#test[1,]
#P(c=Dems|X)
prodVec<-c()
for(i in 1:340){
if(test[1,i]==1){prodVec<-c(prodVec,demVec[i])}else{prodVec<-c(prodVec,(1-demVec[i]))}
}
prior[1]*prod(prodVec)

##   Democratic
## 1.317937e-84

#P(c=Rep|X)
prodVec<-c()
for(i in 1:340){
  if(test[1,i]==1){prodVec<-c(prodVec,repVec[i])}else{prodVec<-c(prodVec,(1-repVec[i]))}
}
prior[2]*prod(prodVec)

##   Republican
## 4.713879e-86

#pridiction : Democratic (Correct !)
```

2. William Clinton 1995

```
#test[2,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:340){
if(test[2,i]==1){prodVec<-c(prodVec,demVec[i])}else{prodVec<-c(prodVec,(1-demVec[i]))}
}
prior[1]*prod(prodVec)

##  Democratic
## 2.57437e-90

#P(c=Rep|X)
prodVec<-c()
for(i in 1:340){
  if(test[2,i]==1){prodVec<-c(prodVec,repVec[i])}else{prodVec<-c(prodVec,(1-repVec[i]))}
}
prior[2]*prod(prodVec)

##   Republican
## 2.618436e-94

# prediction : Democratic (Correct!)
```

3. George w. bush 2006

```
#test[3,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:340){
if(test[3,i]==1){prodVec<-c(prodVec,demVec[i])}else{prodVec<-c(prodVec,(1-demVec[i]))}
}
prior[1]*prod(prodVec)

##    Democratic
## 9.431726e-102

#P(c=Rep|X)
prodVec<-c()
for(i in 1:340){
  if(test[3,i]==1){prodVec<-c(prodVec,repVec[i])}else{prodVec<-c(prodVec,(1-repVec[i]))}
}
prior[2]*prod(prodVec)

##    Republican
## 5.719135e-107

#Prediction : Democratic (Wrong !)
```

4. John F Kennedy , 1962

```
#test[1,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:340){
if(test[4,i]==1){prodVec<-c(prodVec,demVec[i])}else{prodVec<-c(prodVec,(1-demVec[i]))}
}
prior[1]*prod(prodVec)
```

```
##   Democratic
## 2.647376e-99
```

```
#P(c=Rep|X)
prodVec<-c()
for(i in 1:340){
  if(test[4,i]==1){prodVec<-c(prodVec,repVec[i])}else{prodVec<-c(prodVec,(1-repVec[i]))}
}
prior[2]*prod(prodVec)
```

```
##    Republican
## 1.288451e-103
```

```
#Prediction : Democratic (Correct !)
```

```` ``` ````
````

President	Predicted	Actual
Barack Obama, 2014	Democratic	Democratic
William Clinton, 1995	Democratic	Democratic
George W Bush, 2006	Democratic	Republican
John F Kennedy, 1964	Democratic	Democratic

Table 2. Results using Bernoulli model

Another option for classifying the data is the Naïve Bayes with multinomial data set. Rather than classifying data in as a binomial element being 1 or 0, this model creates a feature vector containing a number representing the number of times a given word is present in vocabulary V[1].

Many of the same calculations as with the Bernoulli method are necessary for the multinomial method, including the total number of documents, and the number of documents of each class. With the multinomial model, it is necessary to use sums and factorials in order to calculate the probability of a class given a particular word. The formula for this calculation is shown in figure 1.

$$\hat{P}(w_t \mid C=k) = \frac{\sum_{i=1}^{N} x_{it} z_{ik}}{\sum_{s=1}^{|V|} \sum_{i=1}^{N} x_{is} z_{ik}},$$

Figure 1. Probability calculation using multinomial method

A Perl script was used to generate the multinomial data set, shown in table 3. Using the code in the file multinomial_model.R, the predictions and actual classification using the multinomial data set are shown in table 4.

work	world	year	years	Party
10	41	24	19	Democratic
12	32	18	12	Republican
18	12	16	21	Democratic
9	21	9	18	Republican
18	8	10	13	Democratic

Table 3. Sample of the multinomial data set

Part I

Task 3.

(d).

```
train<-read.csv("./train/multinomial.csv",header=FALSE)
test<-read.csv("./test/multinomialTest.csv",header=FALSE)


#------------------------calculating prior probablilities----
prior <-table(train[,401])/length(train[,401])

#----------------------calculatring conditional probablities----
trainDem<- subset(train,train[,401]=="democratic")
trainRep<- subset(train,train[,401]=="republican")

allDem<-0
for(i in 1:400){
  allDem<-allDem+sum(trainDem[,i])
}
allDem

## [1] 1677

allRep<-0
for(i in 1:400){
  allRep<-allRep+sum(trainRep[,i])
}
allRep

## [1] 1604

#Calculate conditional probablities with add one laplace smoothing.Laplace smoothing has
been done by adding 1 when calculating the likely hood to prevent a likelyhood probabalit
y to becomig zero. For example if a word does not appear atleast one time in Democratic d
ocuments.

demVec<-c()
repVec<-c()
for(i in 1:400){
  demVec<-c(demVec,(sum(trainDem[,i])+1)/(allDem+400))
  #demVec<-c(demVec,sum(trainDem[,i])/allDem)
}

for(i in 1:400){
  repVec<-c(repVec,(sum(trainRep[,i])+1)/(allRep+400))
  #repVec<-c(repVec,sum(trainRep[,i])/allRep)
}
```

Computing the posterior probabilities for test cases 1. Barak Obama 2014

```
#test[1,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:400){
if(test[1,i] > 0){prodVec<-c(prodVec,demVec[i]^test[1,i])}
}
prior[1]*prod(prodVec)

##   democratic
## 1.524004e-15

#P(c=Rep|X)
prodVec<-c()
for(i in 1:400){
  if(test[1,i]>0){prodVec<-c(prodVec,repVec[i]^test[1,i])}
}
prior[2]*prod(prodVec)

##   republican
## 7.630674e-21

#pridiction : Democratic (Correct !)
```

2. William Clinton 1995

```
#test[2,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:400){
if(test[2,i]>0){prodVec<-c(prodVec,demVec[i]^test[2,i])}
}
prior[1]*prod(prodVec)

##   democratic
## 1.223405e-67

#P(c=Rep|X)
prodVec<-c()
for(i in 1:400){
  if(test[2,i]>0){prodVec<-c(prodVec,repVec[i]^test[2,i])}
}
prior[2]*prod(prodVec)

##   republican
## 2.286924e-84

# prediction : Democratic (Correct!)
```

George w. bush 2006

```
#test[3,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:400){
if(test[3,i]>0){prodVec<-c(prodVec,demVec[i]^test[3,i])}
}
prior[1]*prod(prodVec)

##   democratic
## 1.517208e-76

#P(c=Rep|X)
prodVec<-c()
for(i in 1:400){
  if(test[3,i]>0){prodVec<-c(prodVec,repVec[i]^test[3,i])}
}
prior[2]*prod(prodVec)

##  republican
## 1.60289e-51

#Prediction : Democratic (Wrong !)
```

John F Kennedy , 1962

```
#test[1,]

#P(c=Dems|X)
prodVec<-c()
for(i in 1:400){
if(test[4,i]>0){prodVec<-c(prodVec,demVec[i]^test[4,i])}
}
prior[1]*prod(prodVec)

##   democratic
## 3.813866e-72

#P(c=Rep|X)
prodVec<-c()
for(i in 1:400){
  if(test[4,i]>0){prodVec<-c(prodVec,repVec[i]^test[4,i])}
}
prior[2]*prod(prodVec)

##  republican
## 4.23335e-82

#Prediction : Democratic (Correct !)
```

Part II: Sentiment Classification

The movie reviews were split into a training and testing set. A total of 55 reviews were randomly selected with 26 of them having a negative review and 29 having a positive review. Using the training set, a Perl script was created to develop a bag of words for the positive and the negative reviews. Then, a bash script was used to select the highest frequency words from each bag of words file. The common words were also filtered out to create a vocabulary.

Another Perl script was created to calculate the conditional probabilities using the formula in figure 2. The full scripts can be seen in Appendix B. This formula considers Laplace smoothing to address the 0 frequency occurrence of a word in a given class. For example, the word “good” may not appear in any negative document.

$$P(word|positive) = \frac{freq.of\ "word" + 1}{Total\ number\ of\ pos.words + vocabulary\ length}$$

Figure 2. Conditional probability formula for a given word

Using the conditional probabilities calculated with the formula in figure 7, posterior probabilities of the test reviews were calculated. The script output the following to a file after all of the reviews were classified and are shown in table 5.

File	Reference	Predicted	posProb
./t/movies-1-1050.txt	Negative	Positive	1.88E-10
./t/movies-1-1067.txt	Negative	negative	2.91E-29
./t/movies-1-1069.txt	Negative	negative	4.00E-52
./t/movies-1-1076.txt	Negative	negative	6.23E-24
./t/movies-1-1077.txt	Negative	negative	1.21E-30
./t/movies-1-1078.txt	Negative	negative	8.26E-17
./t/movies-1-1087.txt	Negative	negative	5.93E-05
./t/movies-1-1092.txt	Negative	negative	2.06E-26
./t/movies-1-1107.txt	Negative	negative	3.46E-31
./t/movies-1-1145.txt	Negative	negative	1.24E-42
./t/movies-1-1152.txt	Negative	negative	2.41E-17

./t/movies-1-1200.txt	Negative	negative	8.90E-15
./t/movies-1-1019.txt	Negative	positive	1.25E-17
./t/movies-1-1414.txt	Negative	negative	5.29E-17
./t/movies-1-1432.txt	negative	negative	2.71E-18
./t/movies-1-1437.txt	negative	negative	1.37E-32
./t/movies-1-1452.txt	negative	negative	7.84E-41
./t/movies-1-1464.txt	negative	negative	2.75E-14
./t/movies-1-1472.txt	negative	negative	1.38E-31
./t/movies-1-1503.txt	negative	negative	3.80E-13
./t/movies-1-1605.txt	negative	negative	8.61E-09
./t/movies-1-1628.txt	negative	negative	1.02E-47
./t/movies-1-1632.txt	negative	negative	4.90E-32
./t/movies-1-1633.txt	negative	negative	1.99E-25
./t/movies-1-891.txt	negative	negative	3.75E-39
./t/movies-1-927.txt	negative	negative	1.53E-22
./t/movies-5-17446.txt	positive	positive	0.803279873
./t/movies-5-17448.txt	positive	positive	0.00067044
./t/movies-5-17449.txt	positive	positive	1.81E-27
./t/movies-5-17450.txt	positive	positive	1.07E-19
./t/movies-5-17451.txt	positive	positive	1.12E-27
./t/movies-5-17452.txt	positive	positive	3.14E-34
./t/movies-5-17454.txt	positive	positive	2.75E-05
./t/movies-5-17455.txt	positive	positive	5.37E-07
./t/movies-5-17456.txt	positive	positive	6.07E-45
./t/movies-5-17457.txt	positive	positive	9.52E-44

./t/movies-5-17459.txt	positive	positive	2.73E-22
./t/movies-5-17461.txt	positive	positive	9.11E-20
./t/movies-5-17463.txt	positive	positive	6.53E-06
./t/movies-5-17465.txt	positive	positive	5.96E-54
./t/movies-5-17466.txt	positive	positive	6.50E-25
./t/movies-5-17467.txt	positive	positive	1.27E-24
./t/movies-5-17471.txt	positive	positive	2.41E-90
./t/movies-5-17474.txt	positive	positive	7.44E-42
./t/movies-5-17477.txt	positive	positive	9.80E-15
./t/movies-5-17484.txt	Positive	positive	1.85E-24
./t/movies-5-17485.txt	Positive	negative	1.36E-66
./t/movies-5-17488.txt	Positive	positive	6.95E-20
./t/movies-5-17491.txt	Positive	positive	6.42E-14
./t/movies-5-17492.txt	Positive	positive	3.90E-31
./t/movies-5-17495.txt	Positive	positive	1.52E-07
./t/movies-5-8178.txt	Positive	positive	0.803279873
./t/movies-5-8183.txt	Positive	positive	4.00E-13
./t/movies-5-8185.txt	Positive	negative	6.87E-16
./t/movies-5-8188.txt	Positive	positive	3.07E-08
./t/movies-5-8189.txt	Positive	positive	0.039330018

Table 5. Outcomes for movie sentiment

There are several examples where the classifier fails. Here are three examples.

1. "As a violence director, this is even more insipid than Pulp Fiction."

This is a vaguely negative review. It appears to be sarcastic which is extremely difficult for the computer to figure out with the current method being used.

2. “If you want to see a great movie, this is certainly one. I know a lot of people have an aversion to reading subtitles, but this is one time when you shouldn’t let that bother you. This is a completely captivating film, and it’s considered by many to be even better than the Disney version. I’m a little undecided on that, but it’s a very close race. Rent this one (if you can find it) and you’ll be glad you did, just give it a chance.”

There are positive words such as great, captivating, and glad, along with negative words including aversion, bother, and undecided. Since the classifier considers individual meaning rather than the meaning of a group of words, this can present a problem. The negative words have a higher frequency overall than positive words which would result in this being classified as a negative review.

3. “Too funny for words. Total, utter, silliness and well worth watching.”

This review is very hard even for a human to classify as a positive or negative review, let alone a computer. It’s too vague and there is a mix of positive and negative words.

A variety of metrics were calculated for the data including a confusion matrix shown in table 6.

	Reference	
Prediction	Negative	Positive
Negative	24	2
Positive	2	28

Table 6. Confusion matrix for movie review sentiment

Other metrics that were calculated are shown in table 7.

Metric	Value
Accuracy	0.9286
Kappa	0.8564
Sensitivity	0.9231
Specificity	0.9333
AUC	0.9387

Table 7. Calculated metrics for movie reviews

The ROC was plotted and is shown in figure 3.

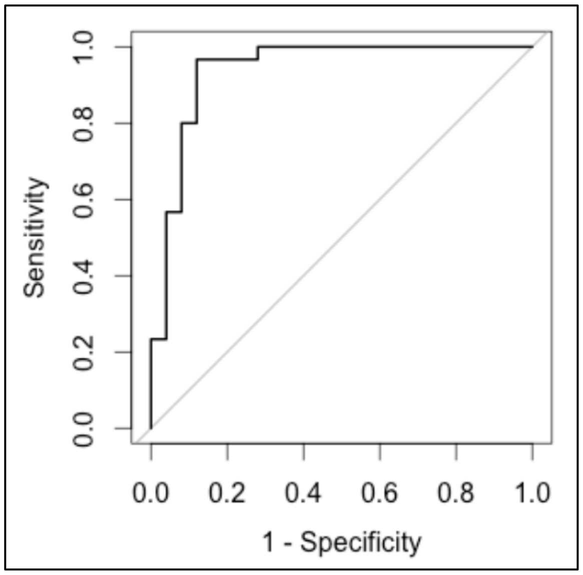

Figure 3. ROC for movie reviews

Part III: Extensions

6) One drawback to the bag of words classifications, using either the Bernoulli model or the multinomial method is the lack of human emotion. While this can be beneficial in letting the data speak for itself without a human intentionally or unintentionally effecting the outcome, it causes problems when looking at the subjectivity of words.

For example, say a person is reviewing a movie. In their review, the person uses the word "sick". This could mean that the person felt the content was disturbing and gross, or it could be meant as a slang term for something great, such as "That is sick!" The classifier would not be able to determine whether to classify this as a positive or a negative review.

This issue has grown as the popularity and prevalence of the internet has grown. There have been many people who have proposed additional methods on how to better classify textual data based on more than just the individual words. One researcher explored selecting key sentences and analyzing these as a method for better evaluating the sentiment of text[2].

Goel looked for three elements when choosing key sentences. First, any sentence that used the movie title was considered key to the review. Second, sentences that used strongly indicative adjectives were selected. Finally, any sentences that used contrast words, for example the word "brilliant" and "terrible" in the same sentence, were given a higher score[2].

These are all excellent methods, although it still isn't a perfect solution. In their research, they were able to improve the accuracy by accounting for these three factors, but this only puts their accuracy in the high 60/low 70 percent accuracy[2].

7) For this assignment, we could have used fuzzy –C means algorithm (FCM). The modified version of this algorithm was used in Netflix Competition'08.

About the data set:
The aim of this competition was to improve Netflix's recommendation system by 10%. The data set contained User IDs and Movie IDs. The user ratings are stored in a vector and averages are taken in the training set and then these ratings are multiplied by a constant called shrink factor of the users and similarly movie ratings are multiplied by shrink factor of the movie. This is called Averaged prediction
Coarse prediction= R_u+R_m-R
Where Ru = shrunk user ratings average
Rm = shrunk movie Ratings average
R=global Ratings

The FCM algorithm can cluster every variable in different classes with suitable probabilities[3]. "This algorithm is based upon classical within groups sum of squared errors objective function":

$$J_m(U, V: X) = \sum_{k=1}^{n}\sum_{i=1}^{c}(u_{ik})^m \|x_k - v_i\|_A^2$$

Figure 4. FCM algorithm[4]

Where m = weighting exponent on each fuzzy membership

U is a fuzzy c-partition on X

V=(v1,v2,v3,.........,v,) are c vector prototypes in R^p;

A = is a positive definite (p x p) matrix

For this assignment, we chose the number of features, or numbers of words possible, to be p=(10,20,30,100……) and assign weight to the documents (w).

Then we created a list of positive keywords from the documents belonging to C_i and then apply the following algorithm:

$$\left|X^d\right| = n_d = 9603\ ,\ \left|X^u\right| = n_u = 3299,\ \ \mathbf{w = weight},\ T = 3000,\ A = I,\ m = 2.0 \text{ and } \varepsilon = 0.01$$

Figure 5 semi supervised FCM algorithm[4]

Accuracy and performance are computed by this algorithm of each partition

Advantages of ssFCM:

- Converges quickly
- Outperforms Naïve Bayes in terms of performance

REFERENCES

1. Shimodaira, H. (2014, February 11). Text Classification using Naive Bayes. Retrieved December 6, 2014, from http://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn-note07-2up.pdf

2. Goel, K., & Hui, A. (2004, June 4). Sentiment Extraction and Classification of Movie Reviews. Retrieved December 6, 2014, from http://nlp.stanford.edu/courses/cs224n/2004/huics_kgoel_final_project.htm

3. Wu Jinlong & Li Tiejun : A Modified Fuzzy C-Means Algorithm For Collaborative Filtering

4. Benkhalifa Mohammed & Bensaid AmineText :Categorization using the Semi-Supervised Fuzzy c-Means Algorithm

APPENDIX A

Pearl Scripts for Part 1

1. Perl script for president's political classification:

```
%presHash=();
open(pres,"pres.txt") or die $!;
while(<pres>){
            chomp;
            my @values=split('\t',$_);
            $values[0] =~ s/^\s+|\s+$//g;
            $values[0] =~ s/\s+//g;
            $key = lc $values[0];
            $presHash{$key}= $values[1];


}


open(LB,"lable1.txt") or die $!;

open(out,">out.txt") or die $!;

$a=0;
while(<LB>){
            $a++;
            chomp;
            my @values2=split(",",$_);


            $values2[0] =~ s/^\s+|\s+$//g;
            $values2[0] =~ s/\s+//g;
            $key = lc $values2[0];
            if(exists($presHash{$key})){
                        print out "a".$a."\t$presHash{$key}\n";
            }
}
```

2. Perl script for Bag of Words

```
open(stop,"stopwords.txt") or die $!;
my %stopWords=();
while(<stop>){
        chomp;
        $stopWords{$_}="1";
}
close(stop);


my $dem="";
my $rep="";
open(file,"dataset.txt") or die $!;

while(<file>){
        chomp;
        my @vals = split("\t",$_);
        if($vals[1] eq "Democratic"){

                $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        unless(exists($stopWords{$word})){ $dem=$dem." ".$word;}

                }
                close(D);

        }else{

                $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        unless(exists($stopWords{$word})){ $rep=$rep." ".$word;}

                }
                close(D);

                }

}



open(bag1,">bagDem.txt") or die $!;


my %wordcount1;
@words = split(' ',$dem);
foreach $word (@words)  {
 $wordcount1{$word}++ if $word =~ /\w/;
}
```

```
foreach $word (keys %wordcount1)  {
 print bag1 "$word\t$wordcount1{$word}\n";
}
close(bag1);
open(bag2,">bagRep.txt") or die $!;

my %wordcount2;
@words = split(' ',$rep);
foreach $word (@words)  {
 $wordcount2{$word}++ if $word =~ /\w/;
}
foreach $word (keys %wordcount2)  {
 print bag2 "$word\t$wordcount2{$word}\n";
}
close(bag2);
```

3. Perl script for Bernoulli data set

```
        my @wordlist;
open(list,"wordlist.txt") or die $!;
                while(<list>){
                        chomp;
                        push(@wordlist,$_);
                }
close(list);

open(final,">bernouli.csv") or die $!;
foreach(@wordlist){print final "$_,";}
print final "party\n";

open(stop,"stopwords.txt") or die $!;
my %stopWords=();
while(<stop>){
        chomp;
        $stopWords{$_}="1";
}
close(stop);


my $dem="";
my $rep="";
open(file,"dataset.txt") or die $!;

while(<file>){
        chomp;
        my %wordHash=();
        my $string;
        my @vals = split("\t",$_);
        if($vals[1] eq "Democratic"){

                $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        unless(exists($stopWords{$word})){ $wordHash{$word}="1";}

                }
                close(D);
                foreach(@wordlist){ if(exists($wordHash{$_})){ print final "1,";}else{print final "0,"}}
                print final "Democratic\n";

        }else{

                $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        unless(exists($stopWords{$word})){$wordHash{$word}="1";}

                }
                close(D);
                foreach(@wordlist){ if(exists($wordHash{$_})){ print final "1,";}else{print final "0,"}}
                print final"Republican\n";
                }
        }
```

4. Script for multinomial model

```
open(stop,"stopwords.txt") or die $!;

my %stopWords=();
while(<stop>){
        chomp;
        $stopWords{$_}="1";
}
close(stop);

my @wordlist;
my %wordlisthash=();

open(list,"wordlist2.txt") or die $!;
while(<list>){
                chomp;
                push(@wordlist,$_);
                $wordlisthash{$_}="1";
        }
close(list);

open(out,">multinomial.csv") or die $!;
open(file,"out.txt") or die $!;

while(<file>){
        chomp;
        my $dem="";
        my $rep="";
        my @vals = split("\t",$_);
        if($vals[1] eq "Democratic"){

                $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        if(!exists($stopWords{$word}) && exists($wordlisthash{$word})){ $dem=$dem."
".$word;}

                }
                close(D);
                #print "Democratic\t$dem\n";
                my %wordcount;
                my @words = split(' ',$dem);
                foreach $word (@words) {
                        $wordcount{$word}++ if $word =~ /\w/;
                }
                foreach $word (@wordlist)  {
                        if(exists($wordcount{$word})){
                        print out "$wordcount{$word},";}else {print out "0,"};
                }
                print out"democratic\n";



        }else{

                 $fileName=$vals[0].".txt";
                open(D,$fileName) or die $!;
                while(<D>){
                        chomp;
                        $word = $_;
                        if(!exists($stopWords{$word}) && exists($wordlisthash{$word})){ $rep=$rep."
".$word;}

                }
                close(D);
                #print "republican\t$rep\n";
                my %wordcount;
                my @words = split(' ',$rep);
                foreach $word (@words) {
                        $wordcount{$word}++ if $word =~ /\w/;
                }
                foreach $word (@wordlist)  {
                        if(exists($wordcount{$word})){
                        print out "$wordcount{$word},";}else {print out "0,"};
                }
                print out "republican\n";


                 }

}
```

APPENDIX B

Pearl Script for Part 2

5. Perl script for movie reviews:

```
open(stop,"stopwords.txt") or die $!;
my %stopWords=();
while(<stop>){
        chomp;
        $stopWords{$_}="1";

}
close(stop);

my $dir = "movies_reviews";
opendir DIR, $dir or die "cannot open dir $dir: $!";
my @files= readdir DIR;
closedir DIR;
my $bad="";
my $good="";
my $goodCount=0;
my $badCount =0;

foreach(@files){
        $filename="./movies_reviews/$_";

        @file = split("-",$_);
        if($file[0] eq "movies"){

                if($file[1] eq "1"){
                        $badCount++;
                        open(file,"$filename") or die $!;
                        $txt = <file>;
                        @values=split(" ",$txt);
                        foreach(@values){
         $_=~ tr/a-zA-Z//dc;
         $_=lc $_;
                        unless(exists($stopWords{$_})){$bad = $bad." ".$_;}
                        }

                }else{
                        $goodCount++;
                        open(file,$filename) or die $!;
                        $txt = <file>;
                        @values=split(" ",$txt);
                        foreach(@values){
         $_=~ tr/a-zA-Z//dc;
         $_=lc $_;
                        unless(exists($stopWords{$_})){$good = $good." ".$_;}
                        }
                }
                close(file);

        }
        }
```

6. Bash script for movie reviews

```
print "good\t$goodCount\n";
print "bad\t$badCount\n";
#
#
open(bag1,">good.txt") or die $!;


my %wordcount1;
@words = split(' ',$good);
foreach $word (@words)  {
 $wordcount1{$word}++ if $word =~ /\w/;
}
foreach $word (keys %wordcount1)  {
 print bag1 "$word\t$wordcount1{$word}\n";
}
close(bag1);
open(bag2,">bad.txt") or die $!;

my %wordcount2;
@words = split(' ',$bad);
foreach $word (@words)  {
 $wordcount2{$word}++ if $word =~ /\w/;
}
foreach $word (keys %wordcount2)  {
 print bag2 "$word\t$wordcount2{$word}\n";
}
close(bag2);

#`cut -f 1 voca.txt | sort -u > vocabulary.txt`;
```

```perl
 no warnings "all";

my %good=();
my %goodProb =();
open(good,"good.txt") or die $!;

while(<good>){
        chomp;
        @values=split("\t",$_);
        $good{$values[0]}=$values[1];
}
close(good);

my %bad=();
my %badProb=();
open(bad,"bad.txt") or die $!;

while(<bad>){
        chomp;
        @values=split("\t",$_);
        $bad{$values[0]}=$values[1];
}
close(bad);


my %voca=();
open(voca,"vocabulary.txt") or die $!;

while(<voca>){
        chomp;
        my $goodProb=0;
        if(exists($good{$_})){
                #print "$good{$_}\n";
                $goodProb=($good{$_})/(216178+685);
                #print "$_\tGood\t$goodProb\n";
                $goodProb{$_}=$goodProb;
        }else{
        $goodProb=(1)/(216178+685);                 # Add one smoothing
                #print "$_\tGood\t$goodProb\n";
                $goodProb{$_}=$goodProb;
        }

        my $badProb=0;
        if(exists($bad{$_})){
                #print "$bad{$_}\n";
                $badProb=($bad{$_})/(44575+685);
                #print "$_\tbad\t$badProb\n";
                $badProb{$_}=$badProb;
        }else{
        $badProb=(1)/(44575+685);                 #Add one smoothing
```

```
                #print "$_\tbad\t$badProb\n";
                $badProb{$_}=$badProb;
        }


}
close(voca);


$isGood=11119/13842;
$isBad=2723/13842;
my $dir = "test";
opendir DIR, $dir or die "cannot open dir $dir: $!";
my @files= readdir DIR;
closedir DIR;

print "File\tACTUAL\tPREDICTED\tposProb\n";
foreach(@files){
    $isGood=11119/13842;
    $isBad=2723/13842;
        $filename="./test/$_";


        @file = split("-",$_);
        if($file[0] eq "movies" & $file[1] eq "1"){
                print "$filename\t";
                print "negative\t";

                open(file,"$filename") or die $!;
                $review=<file>;
                @words=split(' ',$review);
                        foreach(@words){
                $_=~ tr/a-zA-Z//dc;
                $_=lc $_;
                if(exists($goodProb{$_})){$isGood = $isGood* $goodProb{$_};}
                if(exists($badProb{$_})){$isBad = $isBad *$badProb{$_};}
                        }

                        #print "$isGood\n";
                        #print "$isBad\n";

                        if($isGood > $isBad){
                                print "positive\t";

                        }else{
                                print "negative\t";}

                close(file);
        print "$isGood\n";
```

```
            }elsif($file[0] eq "movies" & $file[1] eq "5"){
        $isGood=11119/13842;
        $isBad=2723/13842;
            print "$filename\t";
            print "positive\t";

            open(file,"$filename") or die $!;
            $review=<file>;
            @words=split(' ',$review);
            foreach(@words){
            $_=~ tr/a-zA-Z//dc;
            $_=lc $_;
            if(exists($goodProb{$_})){$isGood = $isGood* $goodProb{$_};}
            if(exists($badProb{$_})){$isBad = $isBad *$badProb{$_};}
                            }

        #print "$isGood\t";
        #print "$isBad\n";

            if($isGood > $isBad){
                    print "positive\t";

            }else{
                    print "negative\t";
    }

                    print "$isGood\n";

                    }
    }
```