NLP Final Project

Ross Ring-Jarvi

4/30/2020

Introduction In this report we will explore natural language processing and the ability to use article text to itentify groups. We will be using the 20 newsgroups dataset comprises of 18846 articles classified into 20 topics. Table 1 shows the articles topics and number of counts for each topic. We will pre-processing the articles into a numerical representation, then using principal component anayasis to visualize the data. We will then split the data into a train/split and train supervised and unsupervised clusting techniques and then test the model on the test data to quantify the accuracy of the model.

Pre-Processing The raw data we have is text from artiles on 20 different topics, an example from 'sci.electronics' group is below.

'From: tell@cs.unc.edu (Stephen Tell)\nSubject: Re: subliminal message flashing on TV\nOrganization: The University of North Carolina at Chapel Hill\nLines: 25\nNNTP-Posting-Host: rukbat.cs.unc.edu\n\nIn article <7480237@hpfcso.FC.HP.COM> myers@hpfcso.FC.HP.COM (Bob Myers) writes:\n>> Hi. I was doing research on subliminal suggestion for a psychology\n>> paper, and I read that one researcher flashed hidden messages on the\n>> TV screen at 1/200ths of a second. Is that possible?\n\n> Might\n>even be a vector ("strokewriter") display, in which case the lower limit\n>on image time is anyone\'s guess (and is probably phosphor-persistence limited).\n\nBack in high school I worked as a lab assistant for a bunch of experimental\npsychologists at Bell Labs. When they were doing visual perception and\nmemory experiments, they used vector-type displays, with 1-millisecond\nrefresh rates common.\n\nSo your case of 1/200th sec is quite practical, and the experimenters were\nprobably sure that it was 5 milliseconds, not 4 or 6 either.\n\n>Bob Myers KC0EW >myers@fc.hp.com \n\nSteve\n-- \nSteve Tell tell@cs.unc.edu H: 919 968 1792 | #5L Estes Park apts\nUNC Chapel Hill Computer Science W: 919 962 1845 | Carrboro NC 27510\nEngineering is a _lot_ like art: Some circuits are like lyric poems, some\nare like army manuals, and some are like The Hitchhiker\'s Guide to the Galaxy..\n'

Our approach to capture the essense of the article is to focus on the words that are unique to the article. We will do this by remove headers, footers, quotes, stop words, and punctuation. Stop words are common words such as "and", "a", "about", "did", "for", "he", "her" etc. Using python package spacy we can remove the unwanted items and retain the unique words to the articles. After apllying our cleaning the 'sci.electronics' example article becomes,

$'highschoolworkedlabassistantbunchexperimentalpsychologistsBellLabsvisualperceptionmemoryexperimentsvectortypedisplays1 - millisecondrefreshratescommoncase1/200thsecpracticalexperimentersprobablysure5milliseconds46Steve'$

Now that we have the cleaned the articles we need to give numerical value to those articles. Python's spacy uses pre-trained vectors for words that then can be combined so that our articles will now be represented as $(300, 1)$. To give a better idea of what spacy's vectorizor does we will give an example of spacy's similarity measure between words. If we take $word1 = cat$ and $word2 = dog$ we can vectorize those words and using spacy's similairy measure we get a similarity meausre of $.8167$ where if we change $word2 = dishwasher$ then we get a similariy measure of $.1834$. Now for each article we have a $(300, 1)$ vector representing the article.

Visulizing the Data After preprocessing we are now working with 300 dimentinal data and to visualize the data we will take the first 2 and 3 principal components of the data to then be able to visualize the data. We can also reduce the deimentaility of the data using PCA to then increase the computational time of fitting models. We will test accuracy of the full data set compared with X number of prinical components which represent X amount of variabily of the data.

Load in NLP data

```
X_train = as.data.frame(read.csv("/Users/rrj/Documents/School/Multivariate/X_train.csv",header = FALSE))  # read
csv file
X_test = as.data.frame(read.csv("/Users/rrj/Documents/School/Multivariate/X_test.csv",header = FALSE) ) # read cs
v file
y_train = as.data.frame(read.csv("/Users/rrj/Documents/School/Multivariate/y_train.csv",header = FALSE))  # read
csv file
y_test = as.data.frame(read.csv("/Users/rrj/Documents/School/Multivariate/y_test.csv",header = FALSE) ) # read cs
v file
X_data=rbind(X_train,X_test)
y_data=rbind(y_train,y_test)
```

PCA

```
n=nrow(X_data)
sts=(n-1)*cov(X_data)
PC=eigen(sts)$vectors
```

```
library(ggplot2)
z=as.matrix(X_data) %*%as.matrix(PC[,1:2])
c=factor(as.matrix(y_data))
sp<-ggplot(as.data.frame(z), aes(z[,1],z[,2] )) + geom_point(aes(color=c))+labs(title = "First 2 Principal Compon
ents of Article data", y="1st Principal Component", x="2nd Principal Component")
sp
```


```
#try 3D
```

Note: Colors 0-19 are assiciated with the Aricle Topics in Table 1.

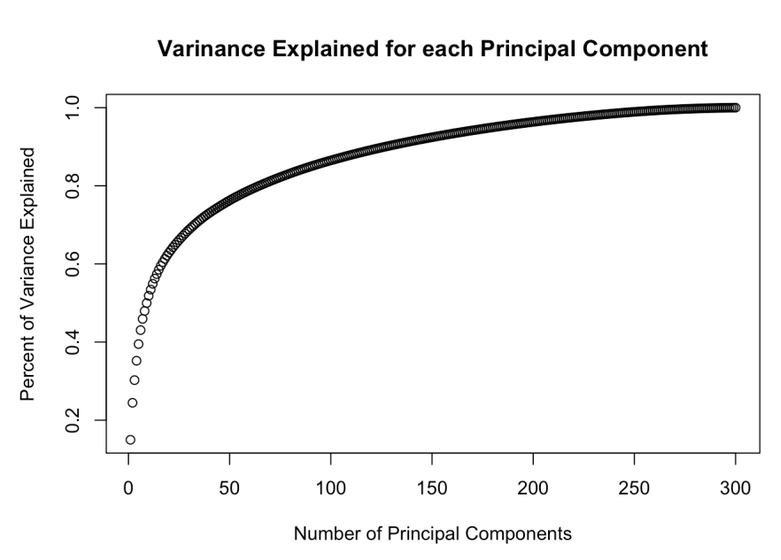
```
totvar = sum(eigen(sts)$values)
cat("Total variance: ",totvar)
```

```
## Total variance:  64706.15
```

```
prop = eigen(sts)$values/totvar
cat("\nProportion of variance explained for first 50 components :",sum(prop[1:50]))
```

```
##
## Proportion of variance explained for first 50 components : 0.7627215
```

```
plot(cumsum(prop), main = "Varinance Explained for each Principal Component", xlab="Number of Principal Component
s", ylab = "Percent of Variance Explained")
```



Create DF with Groups and Counts, counts in train test split

```
z50=as.matrix(X_data) %*%as.matrix(PC[,1:50])
X_train50=as.matrix(z50[1:14134,])
y_train50=as.matrix(y_data[1:14134,])
X_test50=as.matrix(z50[14135:18846,])
y_test50=as.matrix(y_data[14135:18846,])
tc=c(799, 973, 985, 982, 963, 988, 975, 990, 996, 994, 999, 991, 984, 990, 987, 997, 910, 940, 775, 628)
testc=c(205, 245, 250, 243, 255, 240 ,249, 219, 246, 227, 287, 234, 247, 250, 240, 250 ,211 ,246, 209, 159 )
trainc=c(594, 728, 735, 739, 708, 748, 726, 771, 750, 767, 712, 757, 737, 740, 747, 747, 699, 694, 566, 469)
xr=c('alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc')
xc=c("Article Topics","Total_Counts", "Train_Set_Counts", "Test_Set_Counts")
result= data.frame(xr,tc,trainc,testc)
colnames(result)=xc
result
```

K nearest Neighbors Classification We look at K nearest Neighbors classification as our first approach to classify articles. We will use the numberic vectors for each vector to determine which group each data point belongs based on what the items that are nearest to it. This method is non-parametric and requires no assumptions about distributions. We determine the number of k values the algoritm will use to determine what group the point belongs to. We will compare the accuracy of using PCA and the full data set.

```
library(class)
set.seed(1)
acuracy_rate=rep(0,50)
start_time=Sys.time()
for(i in 1:50){
  cls=knn(X_train50[,1:50], X_test50[,1:50], cl=y_train50[,1],k=i)
  a=table(cls, y_test50)
  acuracy_rate[i]=sum(diag(a))
}
end_time=Sys.time()
total_time=(end_time-start_time)
cat("\n k value with highest accuracy :",which.max(acuracy_rate))
```

```
##
##  k value with highest accuracy : 15
```

```
cat("\n First 50 Principal Components Test set accuracy % :",acuracy_rate[which.max(acuracy_rate)]/4712)
```

```
##
##  First 50 Principal Components Test set accuracy % : 0.5717317
```

```
cat("\n Average compute time for 50 runs with first 50 PC :",(total_time*60)/50)
```

```
##
##  Average compute time for 50 runs with first 50 PC : 2.758254
```

We found that for the first 50 principal components, a k value of 15 nearest neighbors gave us the best accuracy of 57% on the test set with an average compute time for each model run of 2.8 seconds.

```
set.seed(1)
start_time=Sys.time()
cls=knn(X_train[,1:300], X_test, cl=y_train[,1],k=15)
end_time=Sys.time()
y1=as.matrix(y_test)
b=table(cls, y1)
cat("\n Full Test set accuracy % :",sum(diag(b))/4712)
```

```
##
##  Full Test set accuracy % : 0.5872241
```

```
cat("\n Average compute time for full data set :",end_time-start_time)
```

```
##
##  Average compute time for full data set : 37.97868
```

Using the full data set we had an accuracy of 58% on the test set with a computation time of 41 seconds. Decreasing the dimensionality from 300 to 50, we lost 1% of accuracy but gain over a 1000% of compute time speed up.

Clusting using Support Vector Machine Support Vector Machines (SVM) attmepts to classify data using vectors(not nessesary linear) to maximize the margin between classes of data. An assumtion we must know is the catagory in which each data point belongs as well as we assume that items in each group are closely together. In 2 dimensions those vectors are lines but in 300 dimentions they are hyperplanes and could be non-linear seperation using kernels. The article data is non-seperable from what we can see in the principal componnets plots thus we will miss-classify multiple data points and we can control the ridgity of the support vector with the cost paramater which we will tune using a linear SVM. We can also use non-linear boundaries using kernels to create a feature mapping of the data and attempt to seperate the data in the feature map. A nice property of kernels are that we do not need to comput the feature mapping and we can use the kernel matrix directly, know as the kernel trick. We will look at radial kernel and attempt to tune the SVM with varity of cost functions and gamma.

```
library(e1071)
svmdf50=as.data.frame(X_train50[,1:50])
svmdf50$Group=y_train50[,1]
svmdf50$Group=as.factor(svmdf50$Group)

start_time=Sys.time()
svmfit=svm(Group~., data=svmdf50, kernel="linear", cost=.01)


#Find predicted classes
pre_01 = predict(svmfit, X_test50[,1:50])
end_time=Sys.time()

y1=as.matrix(y_test)
pre=as.numeric(as.matrix(pre_01))
#table(pre, y1)
cat("\n Cost=.01,1st 50 PC's set accuracy % :",sum(diag(table(pre, y1)))/4712)
```

```
##
##  Cost=.01,1st 50 PC's set accuracy % : 0.6103565
```

```
cat("\n Cost=.01,Compute time for 1st 50 PC's data set :",end_time-start_time)
```

```
##
##  Cost=.01,Compute time for 1st 50 PC's data set : 35.19969
```

```
#Compute the confusion matrix
```

```
library(e1071)
svmdf50=as.data.frame(X_train50[,1:50])
svmdf50$Group=y_train50[,1]
svmdf50$Group=as.factor(svmdf50$Group)

start_time=Sys.time()
svmfit=svm(Group~., data=svmdf50, kernel="polynomial", cost=.0001, gamma=1,degree=3)


#Find predicted classes
pre_01 = predict(svmfit, X_test50[,1:50])
end_time=Sys.time()

y1=as.matrix(y_test)
pre=as.numeric(as.matrix(pre_01))
#table(pre, y1)
cat("\n Cost=.0001,gamma=1,degree3,1st 50 PC's set accuracy % :",sum(diag(table(pre, y1)))/4712)
```

```
##
##  Cost=.0001,gamma=1,degree3,1st 50 PC's set accuracy % : 0.6449491
```

```
cat("\n Cost=.0001,gamma=1,degree3,Compute time for 1st 50 PC's data set :",end_time-start_time)
```

```
##
##  Cost=.0001,gamma=1,degree3,Compute time for 1st 50 PC's data set : 37.85139
```

```
#Compute the confusion matrix
```

```
library(e1071)

svmdf=as.data.frame(X_train[,1:300])
svmdf$Group=y_train[,1]
svmdf$Group=as.factor(svmdf$Group)

start_time=Sys.time()
svmfit=svm(Group~., data=svmdf, kernel="linear", cost=.01)


#Find predicted classes
pre_01 = predict(svmfit, X_test[,1:300])
end_time=Sys.time()

#Compute the confusion matrix
y1=as.matrix(y_test)
pre_01=as.numeric(as.matrix(pre_01))
#table(pre, y1)
cat("\n Cost=.01,Full Test set accuracy % :",sum(diag(table(pre_01, y1)))/4712)
```

```
##
##  Cost=.01,Full Test set accuracy % : 0.6663837
```

```
cat("\n Cost=.01,Compute time for full data set :",(end_time-start_time)*60)
```

```
##
##  Cost=.01,Compute time for full data set : 192.2517
```

```
library(e1071)

svmdf=as.data.frame(X_train[,1:300])
svmdf$Group=y_train[,1]
svmdf$Group=as.factor(svmdf$Group)

start_time=Sys.time()
svmfit=svm(Group~., data=svmdf, kernel="polynomial", cost=.0001, gamma=1,degree=3)


#Find predicted classes
pre_01 = predict(svmfit, X_test[,1:300])
end_time=Sys.time()

#Compute the confusion matrix
y1=as.matrix(y_test)
pre_01=as.numeric(as.matrix(pre_01))
#table(pre, y1)
cat("\n Cost=.0001,gamma=1,degree3,Full Test set accuracy % :",sum(diag(table(pre_01, y1)))/4712)
```

```
##
##  Cost=.0001,gamma=1,degree3,Full Test set accuracy % : 0.667657
```

```
cat("\n Cost=.0001,gamma=1,degree3,Compute time for full data set :",(end_time-start_time)*60)
```

```
##
##  Cost=.0001,gamma=1,degree3,Compute time for full data set : 223.8299
```