Take-home Project: Sentiment Analysis

Ruixuan Zhao MSBA1

2019/12/19

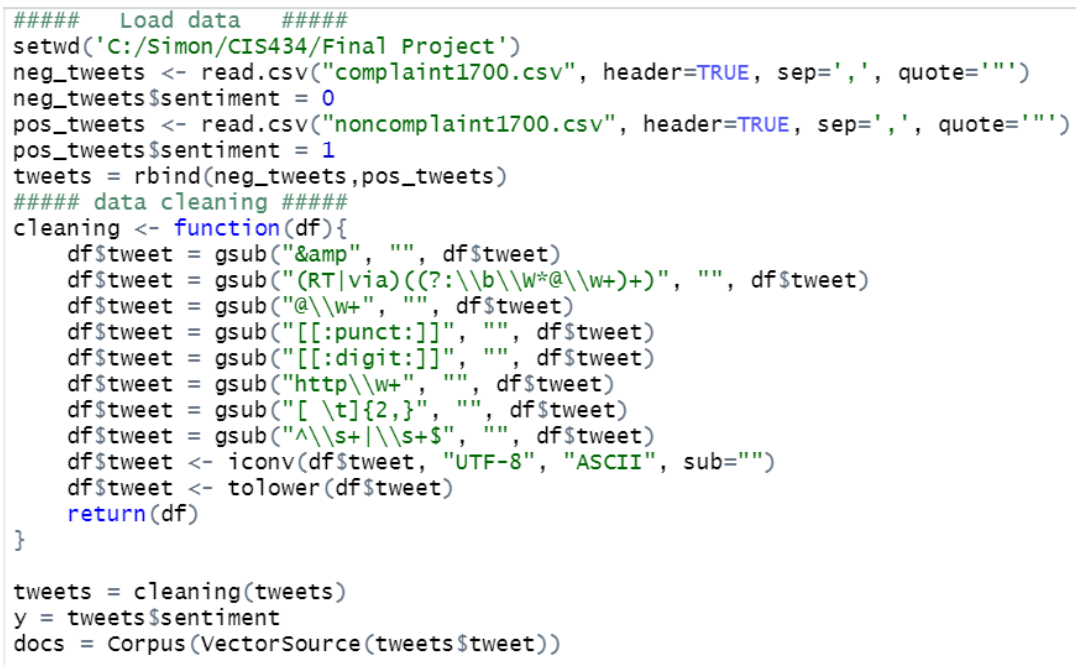
Ruixuan.zhao@simon.rochester.edu

1. Summary

The purpose of this analysis is to train a model to identify the non-complaint tweets from thousands of tweets talking about different airlines. This is a classification problem, so with training dataset complaint1700.csv and noncomplaint1700.csv, I tried three classifying algorithms: svm, rpart, and naïve bayes and finally choose svm to be my classifier. For vectorize the tweets. I tried LDA for topic modeling, SVD for 2-dimension coordinating, and I finally choose DocumentTermMatrix which including 182 terms to vectorize the text data. The cross-validation F1-score(2/(1/precision + 1/recall)) of SVM model is about 0.676. The accuracy of the test dataset from sql database was about 0.65.

2. Details with codes

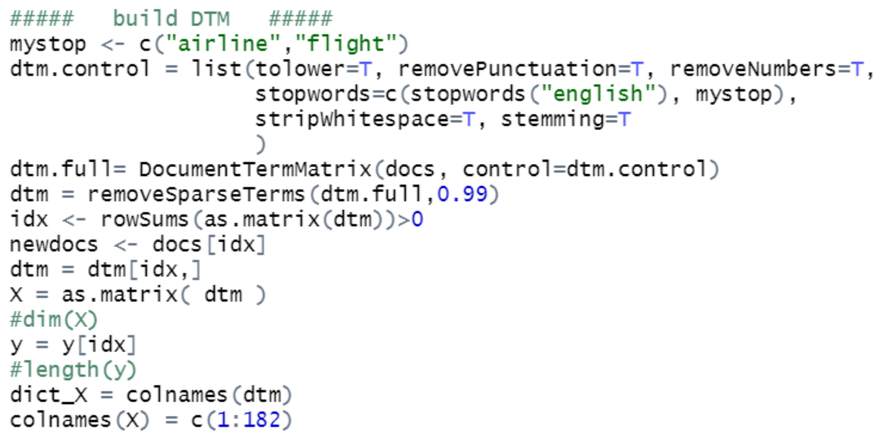
(1) Documents loading and Corpus building



```
#####   Load data   #####
setwd('C:/Simon/CIS434/Final Project')
neg_tweets <- read.csv("complaint1700.csv", header=TRUE, sep=',', quote='"')
neg_tweets$sentiment = 0
pos_tweets <- read.csv("noncomplaint1700.csv", header=TRUE, sep=',', quote='"')
pos_tweets$sentiment = 1
tweets = rbind(neg_tweets,pos_tweets)
##### data cleaning #####
cleaning <- function(df){
    df$tweet = gsub("&amp", "", df$tweet)
    df$tweet = gsub("(RT|via)((?:\\b\\W*@\\w+)+)", "", df$tweet)
    df$tweet = gsub("@\\w+", "", df$tweet)
    df$tweet = gsub("[[:punct:]]", "", df$tweet)
    df$tweet = gsub("[[:digit:]]", "", df$tweet)
    df$tweet = gsub("http\\w+", "", df$tweet)
    df$tweet = gsub("[ \t]{2,}", "", df$tweet)
    df$tweet = gsub("^\\s+|\\s+$", "", df$tweet)
    df$tweet <- iconv(df$tweet, "UTF-8", "ASCII", sub="")
    df$tweet <- tolower(df$tweet)
    return(df)
}

tweets = cleaning(tweets)
y = tweets$sentiment
docs = Corpus(VectorSource(tweets$tweet))
```

The corpus is built up using the text data in complaint1700.csv and noncomplaint1700.csv. I assigned sentiment for both file: complaint-0, noncompliant-1. Then I defined a function called cleaning to clean the text data and I finally got 1999 rows of data for DTM building and forward analysis.

(2) DTM building



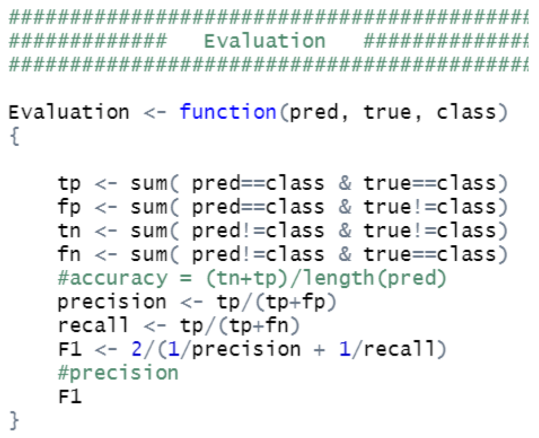
```
#####   build DTM   #####
mystop <- c("airline","flight")
dtm.control = list(tolower=T, removePunctuation=T, removeNumbers=T,
                   stopwords=c(stopwords("english"), mystop),
                   stripWhitespace=T, stemming=T
                   )
dtm.full= DocumentTermMatrix(docs, control=dtm.control)
dtm = removeSparseTerms(dtm.full,0.99)
idx <- rowSums(as.matrix(dtm))>0
newdocs <- docs[idx]
dtm = dtm[idx,]
X = as.matrix( dtm )
#dim(X)
y = y[idx]
#length(y)
dict_X = colnames(dtm)
colnames(X) = c(1:182)
```

After getting the Corpus, I built DTM with “airline” and” flight” to be stopwords and removed sparse term to make the DTM denser and more efficient. Now I get a 1918x182 DTM. Also I created a vector named dict_X, which will used as a dictionary in building DTM on our test dataset, only caring about the terms in the dictionary.

(3) Model training

a) Define evaluation function

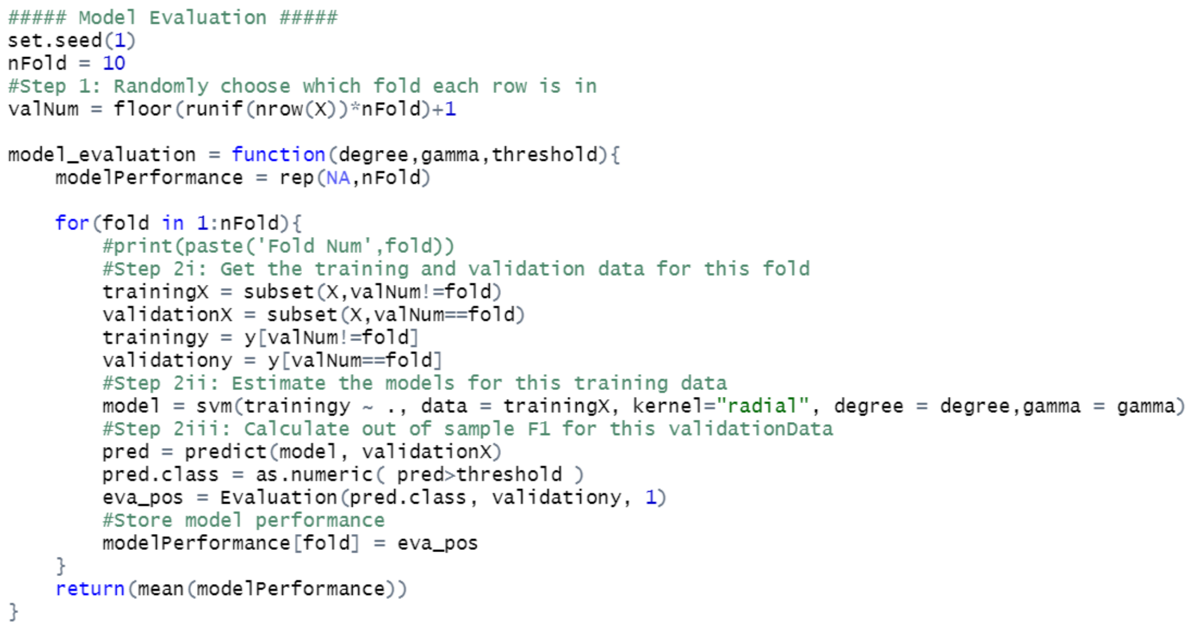
The model evaluation function is used in model training and selecting.



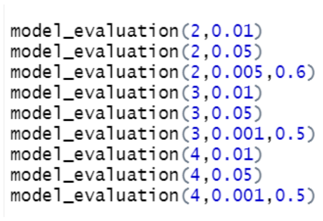
```
############################################
#############   Evaluation   ##############
############################################

Evaluation <- function(pred, true, class)
{

    tp <- sum( pred==class & true==class)
    fp <- sum( pred==class & true!=class)
    tn <- sum( pred!=class & true!=class)
    fn <- sum( pred!=class & true==class)
    #accuracy = (tn+tp)/length(pred)
    precision <- tp/(tp+fp)
    recall <- tp/(tp+fn)
    F1 <- 2/(1/precision + 1/recall)
    #precision
    F1
}
```

b) SVM

The following function below is what I used for tuning the parameter of svm model and calculating the cross-validation F1-score. I used svm with radial kernel, and I tried tuning the degree, gamma and threshold to get my best model.



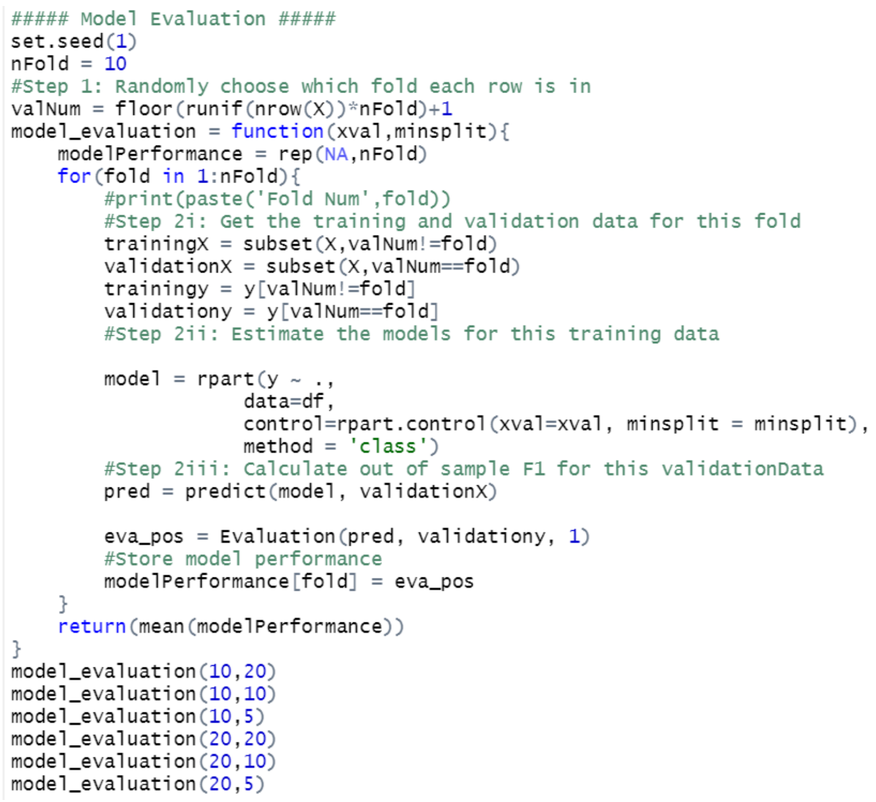
```
##### Model Evaluation #####
set.seed(1)
nFold = 10
#Step 1: Randomly choose which fold each row is in
valNum = floor(runif(nrow(X))*nFold)+1

model_evaluation = function(degree,gamma,threshold){
    modelPerformance = rep(NA,nFold)

    for(fold in 1:nFold){
        #print(paste('Fold Num',fold))
        #Step 2i: Get the training and validation data for this fold
        trainingX = subset(X,valNum!=fold)
        validationX = subset(X,valNum==fold)
        trainingy = y[valNum!=fold]
        validationy = y[valNum==fold]
        #Step 2ii: Estimate the models for this training data
        model = svm(trainingy ~ ., data = trainingX, kernel="radial", degree = degree,gamma = gamma)
        #Step 2iii: Calculate out of sample F1 for this validationData
        pred = predict(model, validationX)
        pred.class = as.numeric( pred>threshold )
        eva_pos = Evaluation(pred.class, validationy, 1)
        #Store model performance
        modelPerformance[fold] = eva_pos
    }
    return(mean(modelPerformance))
}
```



```
model_evaluation(2,0.01)
model_evaluation(2,0.05)
model_evaluation(2,0.005,0.6)
model_evaluation(3,0.01)
model_evaluation(3,0.05)
model_evaluation(3,0.001,0.5)
model_evaluation(4,0.01)
model_evaluation(4,0.05)
model_evaluation(4,0.001,0.5)
```

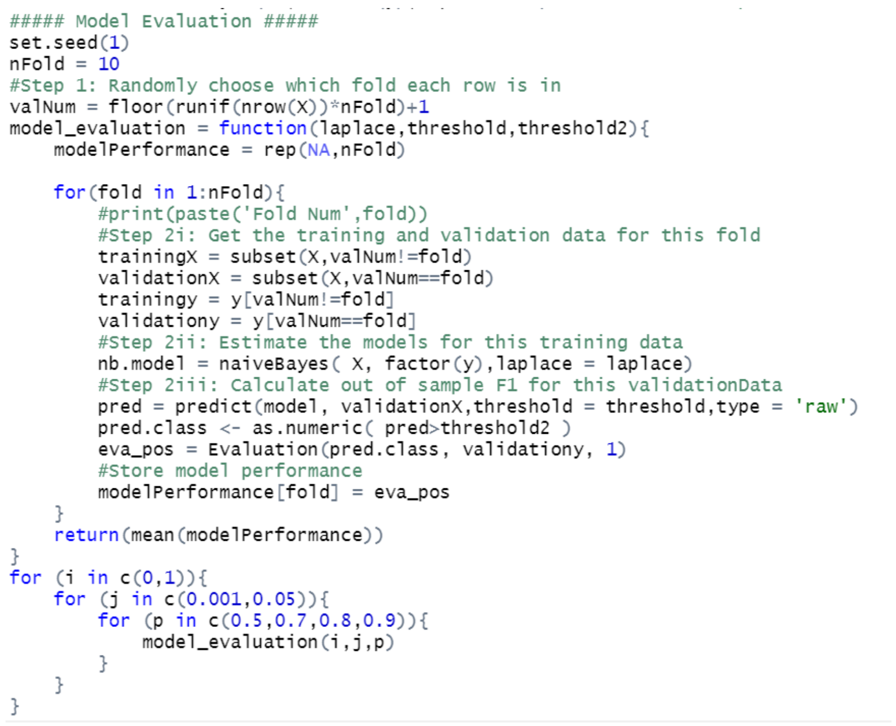
c) Rpart

The same as SVM, the function and process of tuning rpart model are below: the parameter I tuned were xval and minsplit.


```
##### Model Evaluation #####
set.seed(1)
nFold = 10
#Step 1: Randomly choose which fold each row is in
valNum = floor(runif(nrow(X))*nFold)+1
model_evaluation = function(xval,minsplit){
    modelPerformance = rep(NA,nFold)
    for(fold in 1:nFold){
        #print(paste('Fold Num',fold))
        #Step 2i: Get the training and validation data for this fold
        trainingX = subset(X,valNum!=fold)
        validationX = subset(X,valNum==fold)
        trainingy = y[valNum!=fold]
        validationy = y[valNum==fold]
        #Step 2ii: Estimate the models for this training data

        model = rpart(y ~ .,
                      data=df,
                      control=rpart.control(xval=xval, minsplit = minsplit),
                      method = 'class')
        #Step 2iii: Calculate out of sample F1 for this validationData
        pred = predict(model, validationX)

        eva_pos = Evaluation(pred, validationy, 1)
        #Store model performance
        modelPerformance[fold] = eva_pos
    }
    return(mean(modelPerformance))
}
model_evaluation(10,20)
model_evaluation(10,10)
model_evaluation(10,5)
model_evaluation(20,20)
model_evaluation(20,10)
model_evaluation(20,5)
```

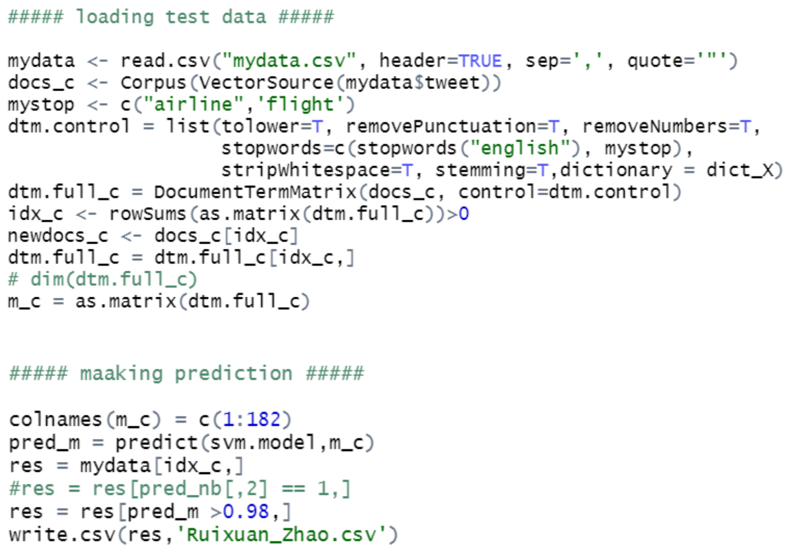
d) Naïve bayes

The same as SVM and rpart, the function and process of tuning rpart model are below:


```
##### Model Evaluation #####
set.seed(1)
nFold = 10
#Step 1: Randomly choose which fold each row is in
valNum = floor(runif(nrow(X))*nFold)+1
model_evaluation = function(laplace,threshold,threshold2){
    modelPerformance = rep(NA,nFold)

    for(fold in 1:nFold){
        #print(paste('Fold Num',fold))
        #Step 2i: Get the training and validation data for this fold
        trainingX = subset(X,valNum!=fold)
        validationX = subset(X,valNum==fold)
        trainingy = y[valNum!=fold]
        validationy = y[valNum==fold]
        #Step 2ii: Estimate the models for this training data
        nb.model = naiveBayes( X, factor(y),laplace = laplace)
        #Step 2iii: Calculate out of sample F1 for this validationData
        pred = predict(model, validationX,threshold = threshold,type = 'raw')
        pred.class <- as.numeric( pred>threshold2 )
        eva_pos = Evaluation(pred.class, validationy, 1)
        #Store model performance
        modelPerformance[fold] = eva_pos
    }
    return(mean(modelPerformance))
}
for (i in c(0,1)){
    for (j in c(0.001,0.05)){
        for (p in c(0.5,0.7,0.8,0.9)){
            model_evaluation(i,j,p)
        }
    }
}
```

e) Testing data

After training and tuning model as shown, the svm with 3-degree radial kernel has the best performance, so I used it for predicting the test data. The result is represented by probability, so I set the threshold to be 0.98 to identify the non-complaint tweets. The result has 190 tweets and I saved it in Ruixuan_Zhao.csv.



```
##### loading test data #####

mydata <- read.csv("mydata.csv", header=TRUE, sep=',', quote='"')
docs_c <- Corpus(VectorSource(mydata$tweet))
mystop <- c("airline",'flight')
dtm.control = list(tolower=T, removePunctuation=T, removeNumbers=T,
                   stopwords=c(stopwords("english"), mystop),
                   stripWhitespace=T, stemming=T,dictionary = dict_X)
dtm.full_c = DocumentTermMatrix(docs_c, control=dtm.control)
idx_c <- rowSums(as.matrix(dtm.full_c))>0
newdocs_c <- docs_c[idx_c]
dtm.full_c = dtm.full_c[idx_c,]
# dim(dtm.full_c)
m_c = as.matrix(dtm.full_c)


##### maaking prediction #####

colnames(m_c) = c(1:182)
pred_m = predict(svm.model,m_c)
res = mydata[idx_c,]
#res = res[pred_nb[,2] == 1,]
res = res[pred_m >0.98,]
write.csv(res,'Ruixuan_Zhao.csv')
```

3. Conclusion

About the project, at the very first beginning, my idea was to use LDA, which is topic modeling. Because DTM is too big and sparse, I was thinking about using LDA to reduce the dimension. Then my thoughts were to calculate the similarity of the tweets in the testing data and those topics based on the definition vector of every topic. However, I failed to calculate a Non-zero matrix that represent the proportion of each test-tweets in each topic. I'm stilling working on it to find a better way. So, I choose DTM after removing the sparse term to train my classifier.

The accuracy of the test data is 0.65, meaning that of those I identify as non-complaining tweets, 59% are not complaining. This result is not good at all. I think the reasons are: 1) training data set are not big enough and not accurate enough; 2) I could have tried more classification model if there are more time; 3) I could have found better way to optimize the DTM with different weight among different terms to help the model to perform better. Though this class is over through this semester, I'll keep working on text mining and text analysis.