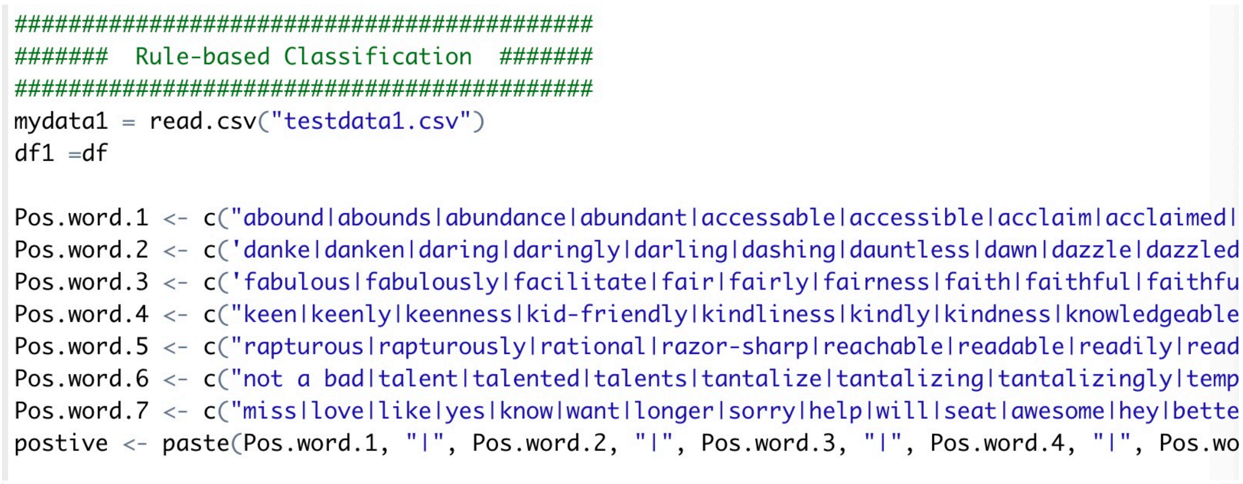
Airline Tweet Analysis Report

Xueqing Hou (MSMA)

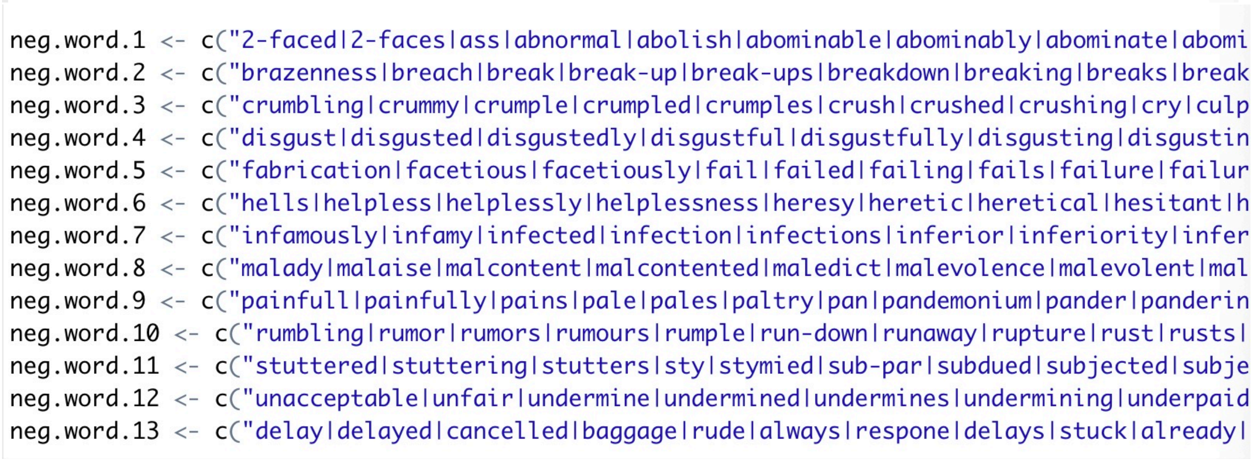
1. Method
   I first used R to analyze the tweet, but the result seems not expected， so I changed to python so that I can use more advanced model. Finally I chose boosting to make an ensemble strong learner with three weak learners: Random Forest, KNN, Naive Bayes.

2. Rule based (R)

a. Insert positive and negative semantic dictionaries:



```
############################################
#######  Rule-based Classification  #######
############################################
mydata1 = read.csv("testdata1.csv")
df1 =df

Pos.word.1 <- c("abound|abounds|abundance|abundant|accessable|accessible|acclaim|acclaimed|
Pos.word.2 <- c('danke|danken|daring|daringly|darling|dashing|dauntless|dawn|dazzle|dazzled
Pos.word.3 <- c('fabulous|fabulously|facilitate|fair|fairly|fairness|faith|faithful|faithfu
Pos.word.4 <- c("keen|keenly|keenness|kid-friendly|kindliness|kindly|kindness|knowledgeable
Pos.word.5 <- c("rapturous|rapturously|rational|razor-sharp|reachable|readable|readily|read
Pos.word.6 <- c("not a bad|talent|talented|talents|tantalize|tantalizing|tantalizingly|temp
Pos.word.7 <- c("miss|love|like|yes|know|want|longer|sorry|help|will|seat|awesome|hey|bette
postive <- paste(Pos.word.1, "|", Pos.word.2, "|", Pos.word.3, "|", Pos.word.4, "|", Pos.wo
```



```
neg.word.1 <- c("2-faced|2-faces|ass|abnormal|abolish|abominable|abominably|abominate|abomi
neg.word.2 <- c("brazenness|breach|break|break-up|break-ups|breakdown|breaking|breaks|break
neg.word.3 <- c("crumbling|crummy|crumple|crumpled|crumples|crush|crushed|crushing|cry|culp
neg.word.4 <- c("disgust|disgusted|disgustedly|disgustful|disgustfully|disgusting|disgustin
neg.word.5 <- c("fabrication|facetious|facetiously|fail|failed|failing|fails|failure|failur
neg.word.6 <- c("hells|helpless|helplessly|helplessness|heresy|heretic|heretical|hesitant|h
neg.word.7 <- c("infamously|infamy|infected|infection|infections|inferior|inferiority|infer
neg.word.8 <- c("malady|malaise|malcontent|malcontented|maledict|malevolence|malevolent|mal
neg.word.9 <- c("painfull|painfully|pains|pale|pales|paltry|pan|pandemonium|pander|panderin
neg.word.10 <- c("rumbling|rumor|rumors|rumours|rumple|run-down|runaway|rupture|rust|rusts|
neg.word.11 <- c("stuttered|stuttering|stutters|sty|stymied|sub-par|subdued|subjected|subje
neg.word.12 <- c("unacceptable|unfair|undermine|undermined|undermines|undermining|underpaid
neg.word.13 <- c("delay|delayed|cancelled|baggage|rude|always|respone|delays|stuck|already|
```

b. Use ‘grepl’ to match the terms in training set.

```r
Neg1 <- paste(neg.word.1, "|", neg.word.2, "|", neg.word.3, "|", neg.word.4, "|", neg.word.5,
Neg2 <- paste(neg.word.7, "|", neg.word.8, "|", neg.word.9, "|", neg.word.10, "|", neg.word.1


df1$pos <- as.numeric(grepl(postive, df1$tweet, ignore.case=TRUE,  perl=TRUE))

dfneg1 <- as.numeric(grepl(Neg1, df1$tweet, ignore.case=TRUE,  perl=TRUE))
dfneg2 <- as.numeric(grepl(Neg2, df1$tweet, ignore.case=TRUE,  perl=TRUE))
df1$neg <- dfneg1 | dfneg2
dfsenti <- df1$pos - df1$neg
df1['predict'] = dfsenti
```

c. Calculate F1 score, F1 score = 0.6.

```r
tp <- sum( df1$predict>=0 & (df1$y==1) )
fp <- sum( df1$predict>=0 & (df1$y==0) )
tn <- sum( df1$predict<0 & (df1$y==0) )
fn <- sum( df1$predict<0 & (df1$y==1) )
precision <- tp/(tp+fp)
recall <- tp/(tp+fn)
score.rule = 2/(1/precision + 1/recall) # F1 score:0.6

mydata1$pos <- as.numeric(grepl(postive, mydata1$tweet, ignore.case=TRUE,  perl=TRUE))

neg1 <- as.numeric(grepl(Neg1, mydata1$tweet, ignore.case=TRUE,  perl=TRUE))
neg2 <- as.numeric(grepl(Neg2, mydata1$tweet, ignore.case=TRUE,  perl=TRUE))
mydata1$neg <- neg1 | neg2
```

d. Use rule based classification to train test set.

```r
mydata1$senti <- mydata1$pos - mydata1$neg

pos_rule = mydata1[mydata1$senti > 0,]
pos_rule = pos_rule[,c(2,4)]
write.csv(pos_rule, file = "rulebase.csv")
```

3. Python

3.1 Clean training data

```python
##Delete the garbled code in the complaint file
def delete(aaa,number):
    #Pass in the original CSV file elements per line and the number of elements
    bbb = []
    for i in range(0,number):
        #Each element in each row is processed separately
        ccc = aaa[i].encode("ASCII","ignore")
        #Delete scrambled code
        bbb.append(ccc.decode())
    return bbb

csv_file1=csv.reader(open('complaint1700.csv','r'))
csv_file=csv.writer(open('complaint_new.csv','a',newline=''),dialect='excel')
content=[]
#The list used to store the entire file,
#stored as a list, each element of the list is another list,
#representing a line of the file

for line in csv_file1:
    number = len(line)
    #The whole CSV file is processed to form a two-dimensional list,
    #in which each element is a list,
    #corresponding to each line of the original CSV file
    #(the scrambled code has been removed).
    content.append(delete(line,number))

for i in range(0,len(content)):
    csv_file.writerow(content[i])
    #Write into a new CSV file
```

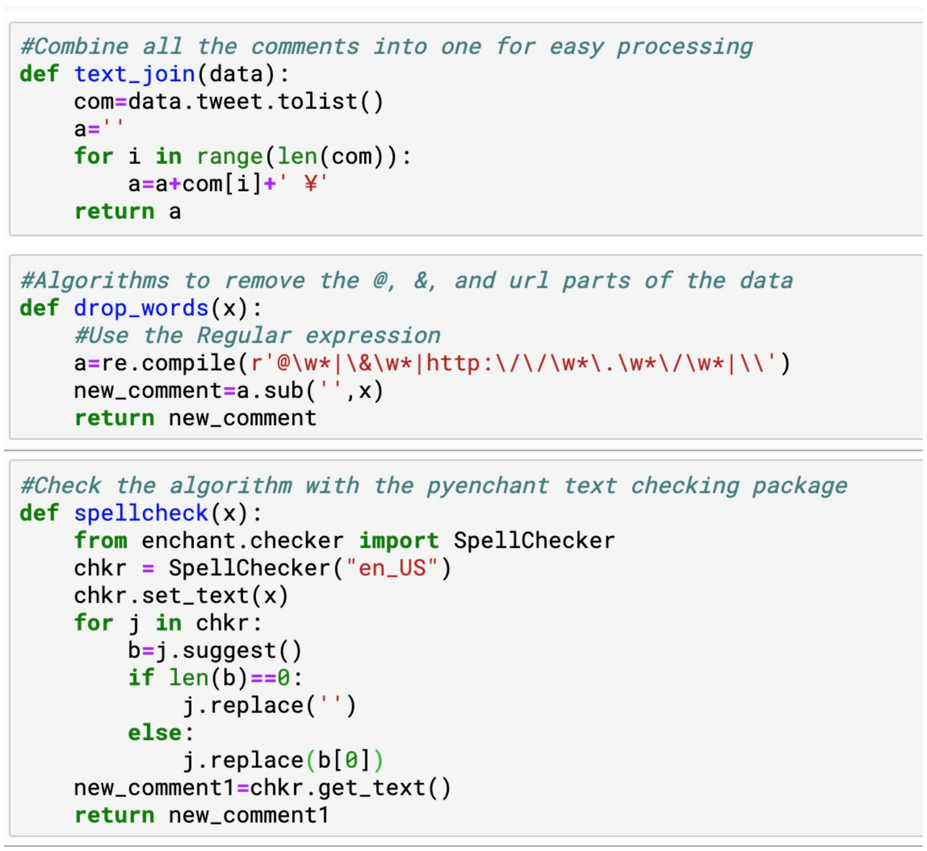
```python
##Delete the garbled code in the noncomplaint file
def delete(aaa,number):
    #Pass in the original CSV file elements per line and the number of elements
    bbb = []
    for i in range(0,number):
        #Each element in each row is processed separately
        ccc = aaa[i].encode("ASCII","ignore")
        #Delete scrambled code
        bbb.append(ccc.decode())
    return bbb

csv_file2=csv.reader(open('noncomplaint1700.csv','r'))
csv_file3=csv.writer(open('noncomplaint_new.csv','a',newline=''),dialect='excel')content=[]
#The list used to store the entire file,
#stored as a list, each element of the list is another list,
#representing a line of the file

for line in csv_file2:
    number = len(line)
     #The whole CSV file is processed to form a two-dimensional list,
    #in which each element is a list,
    #corresponding to each line of the original CSV file
    #(the scrambled code has been removed).
    content.append(delete(line,number))

for i in range(0,len(content)):
    csv_file3.writerow(content[i])
    #Write into a new CSV file
```

```
##Delete the garbled code in the test data file
def delete(aaa,number):
    #Pass in the original CSV file elements per line and the number of elements
    bbb = []
    for i in range(0,number):
        #Each element in each row is processed separately
        ccc = aaa[i].encode("ASCII","ignore")
        #Delete scrambled code
        bbb.append(ccc.decode())
    return bbb

csv_file4=csv.reader(open('testdata1.csv','r'))
csv_file5=csv.writer(open('testdata_new.csv','a',newline=''),dialect='excel')
#The list used to store the entire file,
#stored as a list, each element of the list is another list,
#representing a line of the file

for line in csv_file4:
    number = len(line)
     #The whole CSV file is processed to form a two-dimensional list,
    #in which each element is a list,
    #corresponding to each line of the original CSV file
    #(the scrambled code has been removed).
    content.append(delete(line,number))


for i in range(0,len(content)):
    csv_file5.writerow(content[i])
    #Write into a new CSV file
```

```
complaint=pd.read_csv('complaint_new.csv')
noncomplaint=pd.read_csv('noncomplaint_new.csv')
testdata=pd.read_csv('testdata_new.csv')
```

```
#Combine all the comments into one for easy processing
complaint1=complaint.copy()
noncomplaint1=noncomplaint.copy()
comment=pd.concat([complaint1,noncomplaint1]).reset_index()
#Set the label with 0 for complaint and 1 for non-complaint
target1=np.zeros((complaint1.shape[0],1))
target2=np.ones((noncomplaint1.shape[0],1))
target=np.vstack((target1,target2))
comment['target']=target
```

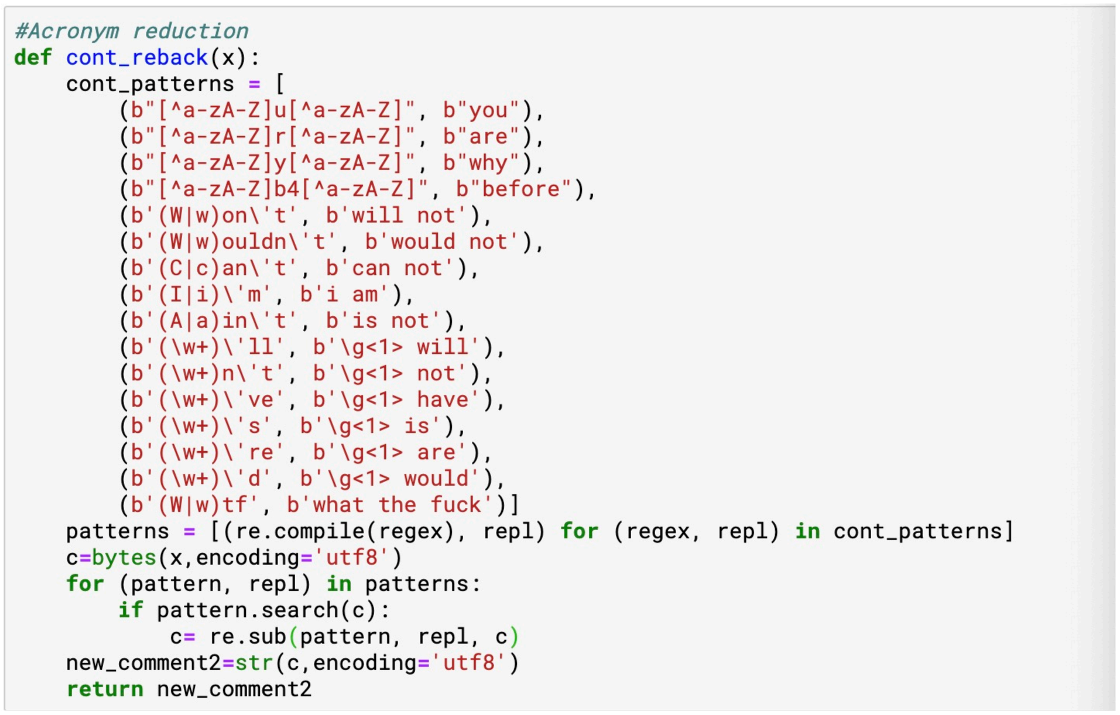
```
comment1=comment.copy()
testdata1=testdata.copy()
```



```python
#Combine all the comments into one for easy processing
def text_join(data):
    com=data.tweet.tolist()
    a=''
    for i in range(len(com)):
        a=a+com[i]+' ¥'
    return a
```

```python
#Algorithms to remove the @, &, and url parts of the data
def drop_words(x):
    #Use the Regular expression
    a=re.compile(r'@\w*|\&\w*|http:\/\/\w*\.\w*\/\w*|\\')
    new_comment=a.sub('',x)
    return new_comment
```

```python
#Check the algorithm with the pyenchant text checking package
def spellcheck(x):
    from enchant.checker import SpellChecker
    chkr = SpellChecker("en_US")
    chkr.set_text(x)
    for j in chkr:
        b=j.suggest()
        if len(b)==0:
            j.replace('')
        else:
            j.replace(b[0])
    new_comment1=chkr.get_text()
    return new_comment1
```



```python
#Acronym reduction
def cont_reback(x):
    cont_patterns = [
        (b"[^a-zA-Z]u[^a-zA-Z]", b"you"),
        (b"[^a-zA-Z]r[^a-zA-Z]", b"are"),
        (b"[^a-zA-Z]y[^a-zA-Z]", b"why"),
        (b"[^a-zA-Z]b4[^a-zA-Z]", b"before"),
        (b'(W|w)on\'t', b'will not'),
        (b'(W|w)ouldn\'t', b'would not'),
        (b'(C|c)an\'t', b'can not'),
        (b'(I|i)\'m', b'i am'),
        (b'(A|a)in\'t', b'is not'),
        (b'(\w+)\'ll', b'\g<1> will'),
        (b'(\w+)n\'t', b'\g<1> not'),
        (b'(\w+)\'ve', b'\g<1> have'),
        (b'(\w+)\'s', b'\g<1> is'),
        (b'(\w+)\'re', b'\g<1> are'),
        (b'(\w+)\'d', b'\g<1> would'),
        (b'(W|w)tf', b'what the fuck')]
    patterns = [(re.compile(regex), repl) for (regex, repl) in cont_patterns]
    c=bytes(x,encoding='utf8')
    for (pattern, repl) in patterns:
        if pattern.search(c):
            c= re.sub(pattern, repl, c)
    new_comment2=str(c,encoding='utf8')
    return new_comment2
```

```python
#Morphological reduction
def wordlemmatize(x):
    d=str.split(x,' ')
    from nltk.stem.wordnet import WordNetLemmatizer
    from nltk import pos_tag
    from nltk import word_tokenize
    def lemmatize(token, tag):
        if tag[0].lower() in ['n', 'v']:
            return lemmatizer.lemmatize(token, tag[0].lower())
        return token
    lemmatizer = WordNetLemmatizer()
    tagged_corpus = [pos_tag(word_tokenize(document)) for document in d]
    e=[]
    for i in tagged_corpus:
        for token, tag in i:
            f=lemmatize(token, tag)
            e.append(f)
    new_comment3=' '.join(e)
    return new_comment3
```

```python
#emove punctuation marks and Numbers
def exclude_word(x):
    exclude=re.compile(r'\.|\,|\;|\?|\$|\d|\\|\/|\\\\|\(|\)|\:|\_|\#')
    new_comment4=exclude.sub('',x)
    return new_comment4
```

```python
#create TF-IDF
def tfidf_demo(x):
    from sklearn.feature_extraction.text import TfidfVectorizer
    #from nltk.corpus import stopwords
    stopwords=['i','you','we','he','she',
               'be','have','they','just','one',
               'the','a','aa','aaa','will',
               'would','as','to','this','there',
               'can','could','of','for']
    tf_idf=TfidfVectorizer(stop_words=stopwords)
    tfidf_matrix = tf_idf.fit_transform(x)
    return tfidf_matrix.toarray(),tf_idf.get_feature_names(),tf_idf
```

```python
def data_preprocessing(data):
    comment=text_join(data)
    new_comment=drop_words(comment)
    new_comment1=spellcheck(new_comment)
    new_comment2=cont_reback(new_comment1)
    new_comment3=wordlemmatize(new_comment2)
    new_comment3=new_comment3.lower()
    new_comment4=exclude_word(new_comment3)
    comment_new=str.split(new_comment4,'¥')
    feature_matrix,feature_names,tf_idf=tfidf_demo(comment_new)

    return feature_matrix,feature_names,tf_idf
```

```python
dataset,feature_names,tf_idf=data_preprocessing(comment1)
```

```python
labelset=np.array(list(comment.target.values)+[1])
```

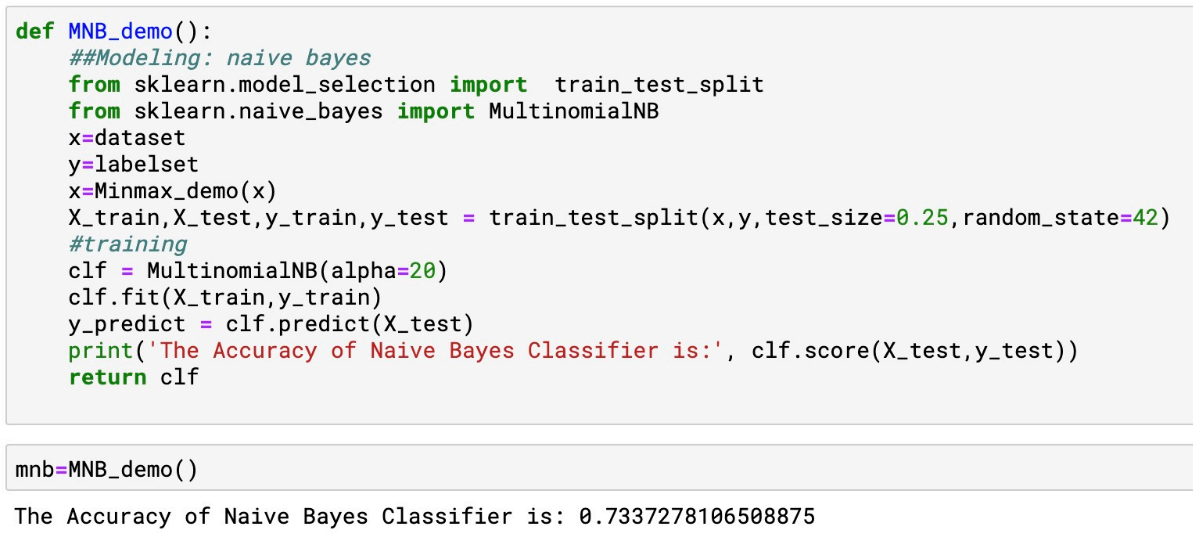
```python
#Standardized
def Standard_demo(x):
    from sklearn.preprocessing import StandardScaler
    standard=StandardScaler(with_mean=False)
    standard_result=standard.fit_transform(x)
    return standard_result
```

```python
#Minmax
def Minmax_demo(x):
    from sklearn.preprocessing import MinMaxScaler
    minmax=MinMaxScaler()
    minmax_result=minmax.fit_transform(x)
    return minmax_result
```

```python
#Dimension reduction
def PCA_demo(x):
    from sklearn.decomposition import PCA
    pca=PCA(n_components=0.95)
    x_pca=pca.fit_transform(x)
    return x_pca
```

3.2 Naive Bayes

a. Maximum and minimum characterization processing
b. Divide the training set and the test set. 75% was selected as the training set, 25% as the test, and 42 random values were added
c. Train model and print accuracy



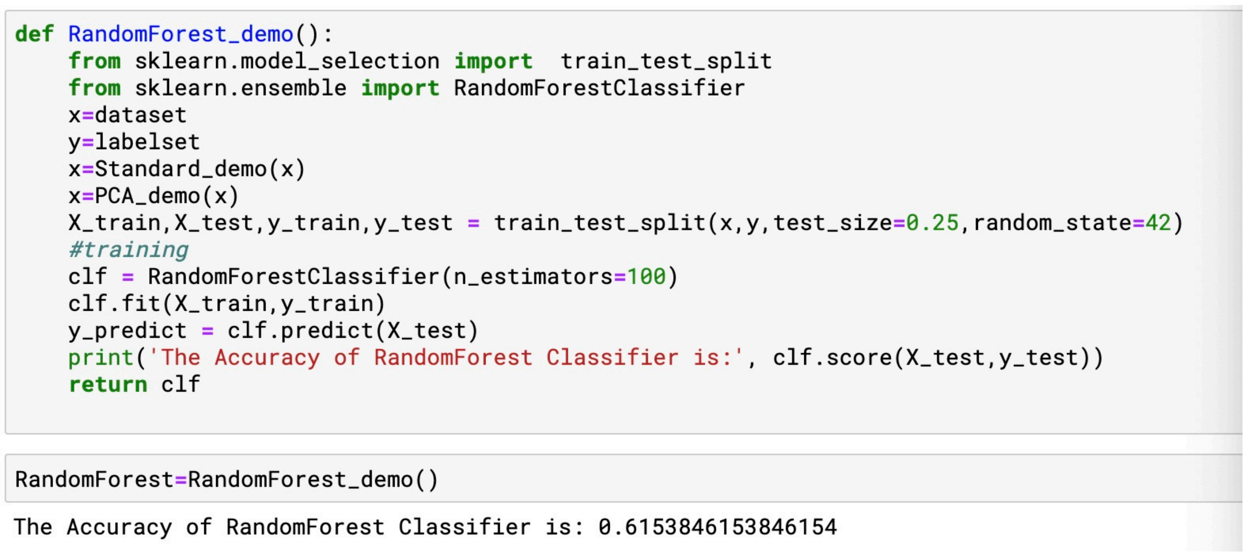
```
def MNB_demo():
    ##Modeling: naive bayes
    from sklearn.model_selection import  train_test_split
    from sklearn.naive_bayes import MultinomialNB
    x=dataset
    y=labelset
    x=Minmax_demo(x)
    X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=42)
    #training
    clf = MultinomialNB(alpha=20)
    clf.fit(X_train,y_train)
    y_predict = clf.predict(X_test)
    print('The Accuracy of Naive Bayes Classifier is:', clf.score(X_test,y_test))
    return clf
```

```
mnb=MNB_demo()
```

```
The Accuracy of Naive Bayes Classifier is: 0.7337278106508875
```

3.3 Random Forest

a. Standardized
b. Principal component analysis was used to reduce dimension
c. Divide the training set and the test set. 75% was selected as the training set, 25% as the test, and 42 random values were added
d. Select 100 small decision trees
e. Train model and print accuracy



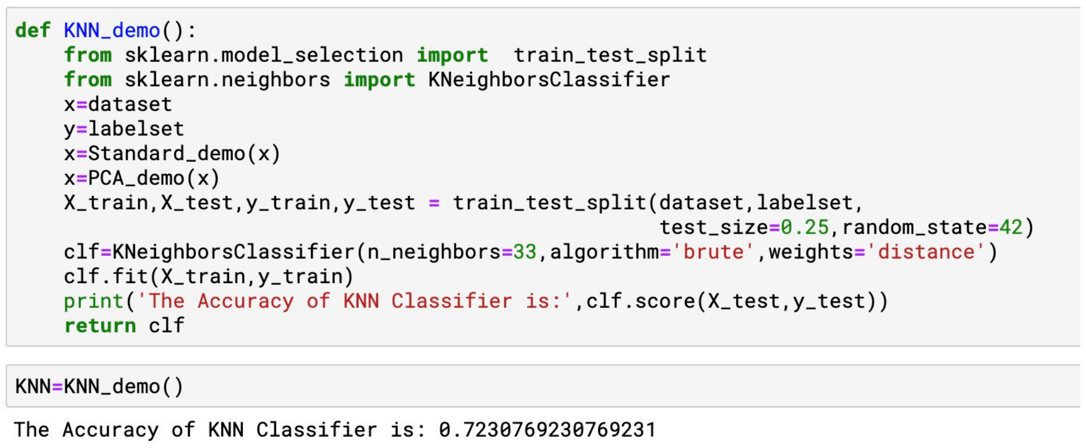
```
def RandomForest_demo():
    from sklearn.model_selection import  train_test_split
    from sklearn.ensemble import RandomForestClassifier
    x=dataset
    y=labelset
    x=Standard_demo(x)
    x=PCA_demo(x)
    X_train,X_test,y_train,y_test = train_test_split(x,y,test_size=0.25,random_state=42)
    #training
    clf = RandomForestClassifier(n_estimators=100)
    clf.fit(X_train,y_train)
    y_predict = clf.predict(X_test)
    print('The Accuracy of RandomForest Classifier is:', clf.score(X_test,y_test))
    return clf
```

```
RandomForest=RandomForest_demo()
```

```
The Accuracy of RandomForest Classifier is: 0.6153846153846154
```

3.4 K Nearest Neighbor

a. Standardized
b. Principal component analysis was used to reduce dimension
c. Divide the training set and the test set. 75% was selected as the training set, 25% as the test, and 42 random values were added
d. Take k=33 (33 nearest neighbors) and calculate its distance by brute force. The distance is weighted by distance.
e. Train model and print accuracy



```
def KNN_demo():
    from sklearn.model_selection import  train_test_split
    from sklearn.neighbors import KNeighborsClassifier
    x=dataset
    y=labelset
    x=Standard_demo(x)
    x=PCA_demo(x)
    X_train,X_test,y_train,y_test = train_test_split(dataset,labelset,
                                                     test_size=0.25,random_state=42)
    clf=KNeighborsClassifier(n_neighbors=33,algorithm='brute',weights='distance')
    clf.fit(X_train,y_train)
    print('The Accuracy of KNN Classifier is:',clf.score(X_test,y_test))
    return clf
```

```
KNN=KNN_demo()
```

```
The Accuracy of KNN Classifier is: 0.7230769230769231
```

3.5 Boosting

a. Standardized
b. Principal component analysis was used to reduce dimension
c. Divide the training set and the test set. 75% was selected as the training set, 25% as the test, and 42 random values were added
d. Select the three classifiers that have just been trained
e. Proportional weighting. The proportion of naive bayes and KNN is 3/7, and the proportion of random forest is 1/7

```
def vote_demo():
    from sklearn.ensemble import VotingClassifier
    from sklearn.model_selection import  train_test_split
    from sklearn.neighbors import KNeighborsClassifier
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.naive_bayes import MultinomialNB

    x=dataset
    y=labelset
    x=Standard_demo(x)
    x=PCA_demo(x)
    X_train,X_test,y_train,y_test = train_test_split(dataset,labelset,
                                                     test_size=0.25,random_state=42)
    clf=VotingClassifier(estimators=
                         [('mnb',mnb),
                          ('RandomForest',RandomForest),
                          ('KNN',KNN)],voting='soft',weights=[3,1,3])
    clf.fit(X_train,y_train)

    print('The Accuracy of voting Classifier is:',clf.score(X_test,y_test))
    return clf
```

```
vote=vote_demo()
```

```
The Accuracy of voting Classifier is: 0.749112426035503
```

3.6 predict test data

```
#cleaning test data
def test_demo():
    comment=text_join(testdata1)
    new_comment=drop_words(comment)
    new_comment1=spellcheck(new_comment)
    new_comment2=cont_reback(new_comment1)
    new_comment3=wordlemmatize(new_comment2)
    new_comment3=new_comment3.lower()
    new_comment4=exclude_word(new_comment3)
    comment_new=str.split(new_comment4,'¥')
    test_tf_idf =tf_idf.transform(comment_new)
    x=test_tf_idf
    x=Standard_demo(x)
    prediction=vote.predict(x)
    return prediction
```

```
#predict test data
prediction=test_demo()
```

```
#choose the noncomplaint tweet and id
testdata1['prediction']=list(prediction)[0:-1]
testdata1=testdata1[testdata1['prediction']==1]
testdata1=testdata1.iloc[:,[1,3]].set_index('id')
```

```
#output
testdata1.to_csv('testdata1_new_complain.csv')
```

4. Conclusion

The boosting model found out 485 tweets. After manually classification, there were about   382 non-complaint tweets, the precision is 0.788.