Social Media Analytics - Final Project

Ruiling Shen (31581004)

1. Loading the data

In [0]:
```
import os
import numpy as np
import pandas as pd
import re
import nltk
import matplotlib.pyplot as plt
import seaborn as sns
import string
import warnings
import emoji # pip install emoji
warnings.filterwarnings("ignore")

from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords # nltk.download('stopwords')
from sklearn.base import BaseEstimator, TransformerMixin

%matplotlib inline
```

In [0]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?
client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuser
content.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_t
ype=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.t
est%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fw
ww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.go
ogleapis.com%2fauth%2fpeopleapi.readonly

Enter your authorization code:
··········
Mounted at /content/drive
```

In [0]:
```
df_complaint = pd.read_csv("/content/drive/My Drive/SIMON /Fall B /CIS43
4 Social Media /Final Project /data/training data/complaint1700.csv")
df_noncomplaint = pd.read_csv("/content/drive/My Drive/SIMON /Fall B /CI
S434 Social Media /Final Project /data/training data/noncomplaint1700.cs
v")
```

In [0]:
```
df_test = pd.read_csv("/content/drive/My Drive/SIMON /Fall B /CIS434 Social Media /Final Project /data/test_data.csv")
df_test.drop(['tid_not_to_be_used','airline','tag'],axis=1)
```

Out[0]:

	id	tweet
0	8	@ArianFoster I hate @united 2. They didn't let...
1	10	Slightly delayed flight home from Philly, but ...
2	15	Hey @AmericanAir you suck.
3	29	The reps of @JetBlue are far less than pleasan...
4	41	@AmericanAir yo your plane peanuts suck ass an...
...	...	...
4550	173592	Hey @SouthwestAir, we're trying to get home to...
4551	173611	@SpiritAirlines flight delayed 5 hours & c...
4552	173618	@united Poor customer service!! My dad lost hi...
4553	173638	So @United schooling flight attendants not to ...
4554	173649	@AmericanAir the people working for you FAIL M...

4555 rows × 2 columns

In [0]:
```
df_complaint['tag'] = 'Bad'
df_noncomplaint['tag'] = 'Good'
```

In [0]:
```
df = pd.concat([df_complaint,df_noncomplaint],axis=0)
```

In [0]:
```
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3400 entries, 0 to 1699
Data columns (total 4 columns):
id          3400 non-null int64
airline     3400 non-null object
tweet       3400 non-null object
tag         3400 non-null object
dtypes: int64(1), object(3)
memory usage: 132.8+ KB
None
```

2. Exploratory Data Analysis

2.1 Count Special Elements of Tweets

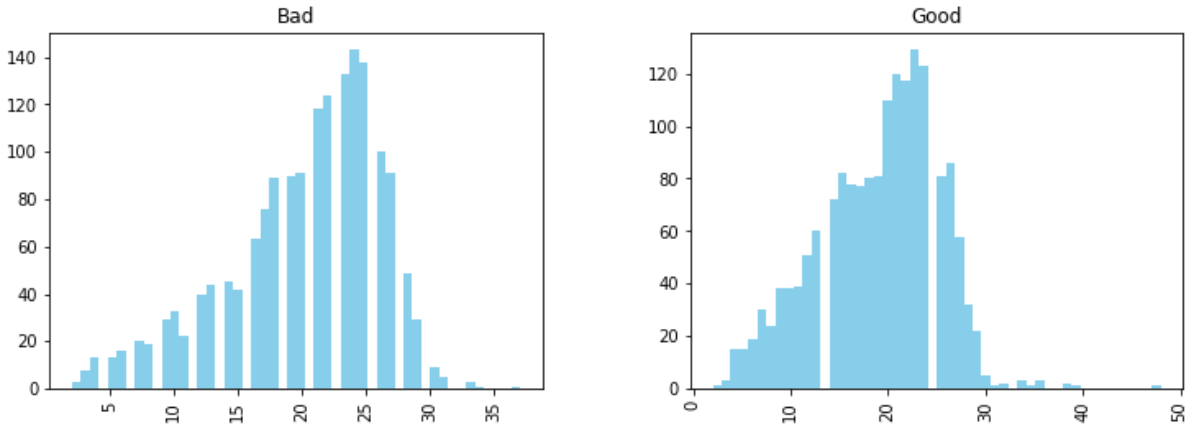
In [0]:
```
class TextCounts(BaseEstimator, TransformerMixin):

    def count_regex(self, pattern, tweet):
        return len(re.findall(pattern, tweet))

    def fit(self, X, y=None, **fit_params):
        # fit method is used when specific operations need to be done on
the train data, but not on the test data
        return self

    def transform(self, X, **transform_params):
        count_words = X.apply(lambda x: self.count_regex(r'\w+', x))
        count_mentions = X.apply(lambda x: self.count_regex(r'@\w+', x))
        count_hashtags = X.apply(lambda x: self.count_regex(r'#\w+', x))
        count_capital_words = X.apply(lambda x: self.count_regex(r'\b[A-
Z]{2,}\b', x))
        count_excl_quest_marks = X.apply(lambda x: self.count_regex(r'!|
\?', x))
        count_urls = X.apply(lambda x: self.count_regex(r'http.?://[^\s]
+[\s]?', x))
        # We will replace the emoji symbols with a description, which ma
kes using a regex for counting easier
        # Moreover, it will result in having more words in the tweet
        count_emojis = X.apply(lambda x: emoji.demojize(x)).apply(lambda
x: self.count_regex(r':[a-z_&]+:', x))

        df = pd.DataFrame({'count_words': count_words
                           , 'count_mentions': count_mentions
                           , 'count_hashtags': count_hashtags
                           , 'count_capital_words': count_capital_words
                           , 'count_excl_quest_marks': count_excl_quest_
marks
                           , 'count_urls': count_urls
                           , 'count_emojis': count_emojis
                          })

        return df
```

In [0]:
```
tc = TextCounts()
df_count = tc.fit_transform(df.tweet)
df = pd.concat([df,df_count],axis=1)
```

In [0]:
```
def get_difference_plot(col):
  return df.hist(column=col,by='tag',bins=50,figsize=(12,4),color='skybl
ue')
```

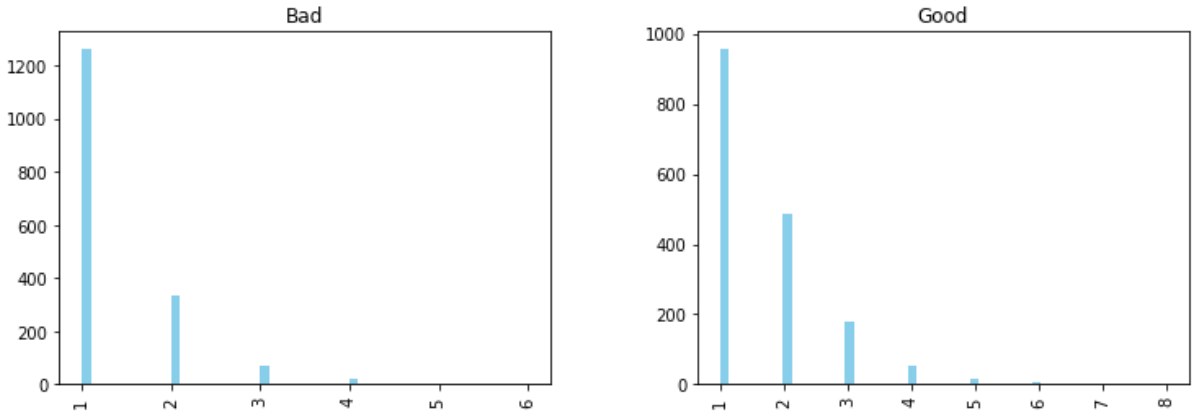
In [0]:
```
get_difference_plot('count_words')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5fc50f0
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5ed2198
>],
      dtype=object)
```



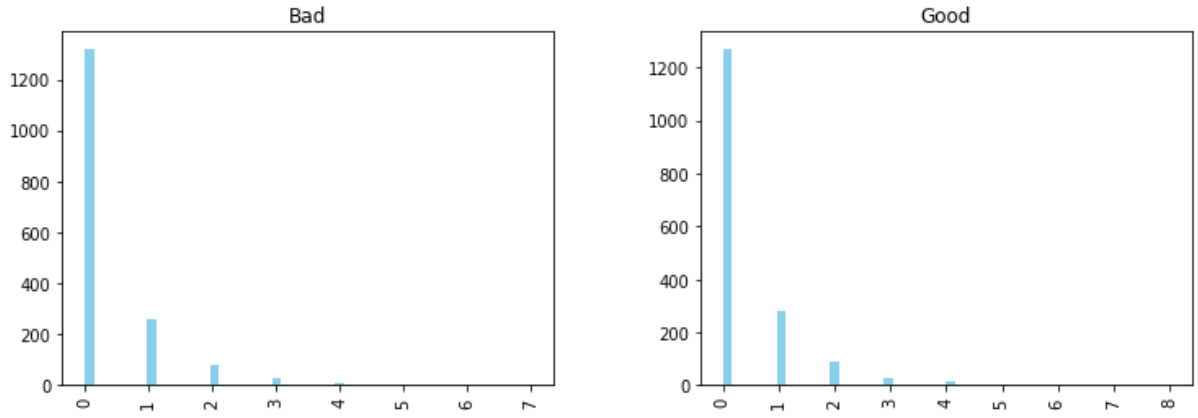
In [0]:
```
get_difference_plot('count_mentions')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5d68898
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5d305f8
>],
      dtype=object)
```



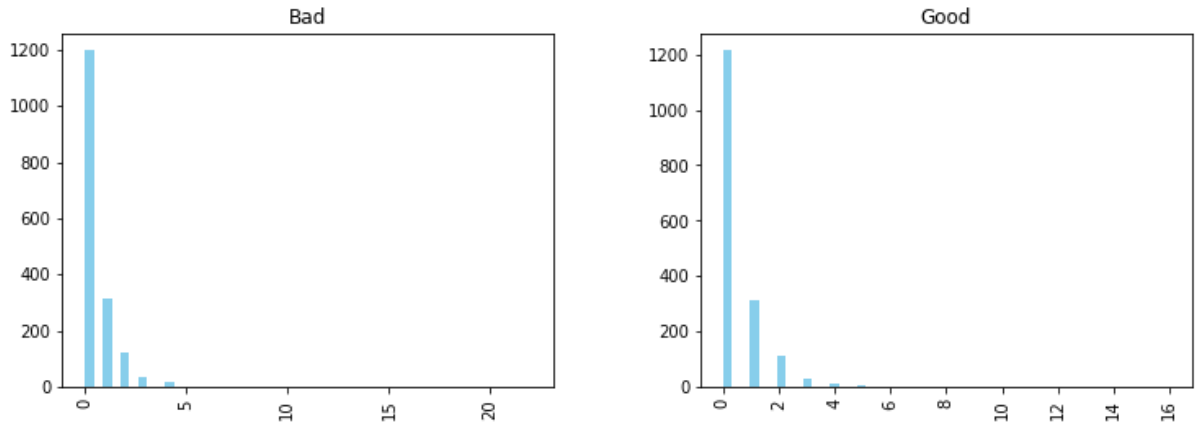
In [0]:
```
get_difference_plot('count_hashtags')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5b419e8
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5b72748
>],
      dtype=object)
```



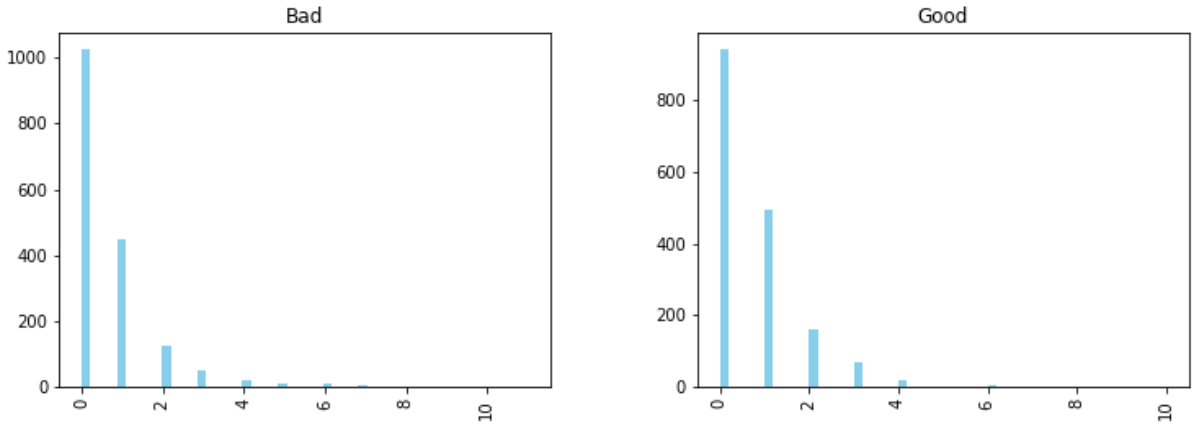
In [0]:
```
get_difference_plot('count_capital_words')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5a5ea20
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb59b9d68
>],
      dtype=object)
```



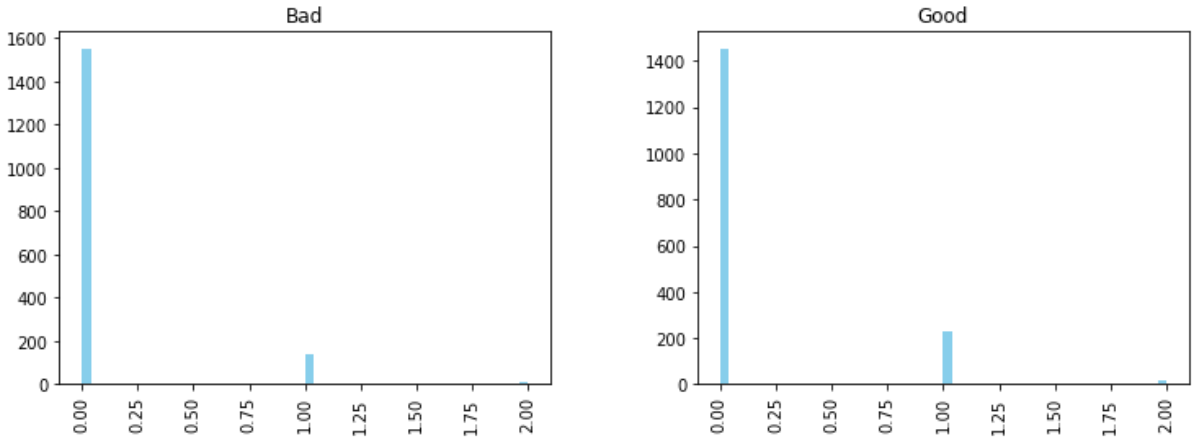
In [0]:
```
get_difference_plot('count_excl_quest_marks')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb57d0f28
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb57ff7b8
>],
      dtype=object)
```



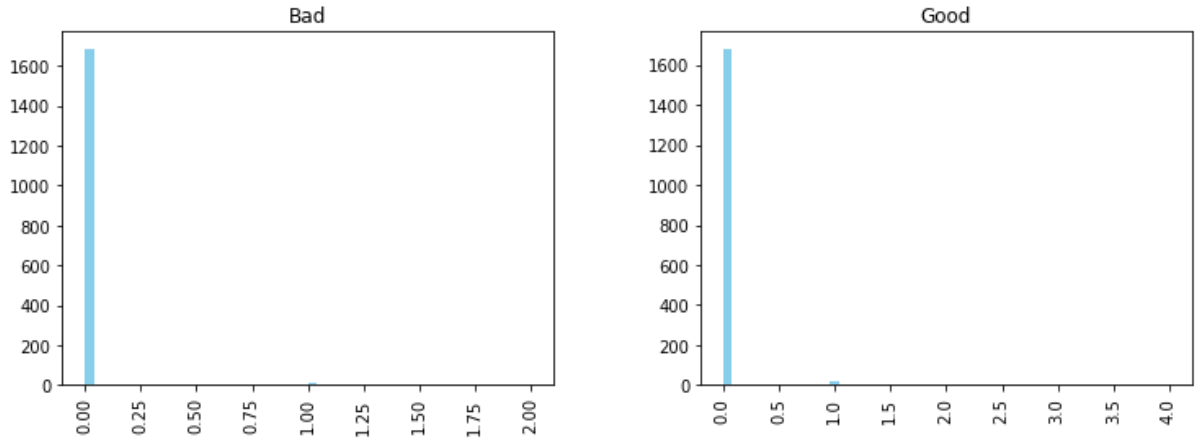
In [0]:
```
get_difference_plot('count_urls')
```

Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5623f60
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb56396d8
>],
      dtype=object)
```



In [0]:
```
get_difference_plot('count_emojis')
```

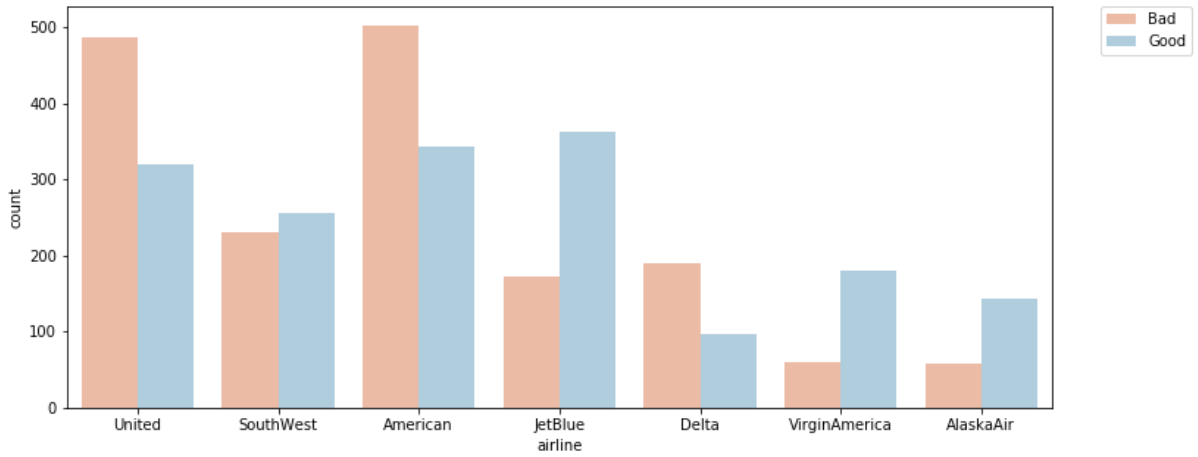
Out[0]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb5448940
>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x7f8cb543cd68
>],
      dtype=object)
```



From those pair of plots, I cannot find a significant difference between complain and non-complain tweets. So I would not add those features into my predicitve analysis.

2.2 To check whether positive and negative tweets varies from airlines.

In [0]:
```
plt.figure(figsize = (12,5))
sns.countplot(x='airline',hue='tag',palette='RdBu',data=df)
plt.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.)
plt.show()
```



From this plot, for some airline it received positive tweets more than negative, such as JetBlue, VirginAmerica, and AlaskaAir. Nevertheless, United, American, and Delta showed an opposite trend.

So airline info should have a potential predictive power on our model.

3. Data Pre-processing

3.1 Text Pre-processing

In order to analyze text on the whole dataset, I will create some function here, and then use data pipeline to apply those function to the whole dataset.

In [0]:
```
def listToString(s):
  """
  This function can make a list to a string variable so that I can fur t
her utilize word or sentence tokenization from nltk package
  """
  str1 = " "
  return (str1.join(s))
```

In [0]:
```
useless_punc = [char for char in string.punctuation if char!='?']
useless_punc
```

Out[0]:
```
['!',
 '"',
 '#',
 '$',
 '%',
 '&',
 "'",
 '(',
 ')',
 '*',
 '+',
 ',',
 '-',
 '.',
 '/',
 ':',
 ';',
 '<',
 '=',
 '>',
 '@',
 '[',
 '\\',
 ']',
 '^',
 '_',
 '`',
 '{',
 '|',
 '}',
 '~']
```

In [0]:
```
def remove_punc_stopwords(text):
  """
  This function performe the following processing steps of a text messag
e:
  1. word tokenize the text message
  2. Remove all the punction, except ? or !
  3. Remove all stopwords
  4. return to a list of the cleaned text
  """
  # word tokenize the text message
  word_list = nltk.word_tokenize(text)

  # check characters to see if they are in punctuation
  not_in_punc = [word for word in word_list if word.lower() not in usele
ss_punc]

  # Now just remove any stopwords
  return [word.lower() for word in not_in_punc if word.lower() not in st
opwords.words(['english','french','spanish','portuguese'])]
```

3.2 Term-Document Matrix

In [0]:
```
from sklearn.feature_extraction.text import CountVectorizer
```

In [0]:
```
# generate the tdm
tdm_transformer = CountVectorizer(analyzer=remove_punc_stopwords).fit(df
['tweet']) # create the model
  # but we havn't tranform the df into tdm yet!
```

In [0]:
```
# transform the df into tdm!
df_tdm = tdm_transformer.transform(df['tweet'])
```

In [0]:
```
print('shape of the TDM:', df_tdm.shape)
```

```
shape of the TDM: (3400, 8465)
```

3.3 TF-IDF

In [0]:
```
from sklearn.feature_extraction.text import TfidfTransformer

tfidf_transformer = TfidfTransformer().fit(df_tdm)
df_tfidf = tfidf_transformer.transform(df_tdm)
```

4. Model Training

4.1 Pipeline & Train-Test-Split

In [0]:
```
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report
from sklearn.model_selection import KFold, cross_val_score, GridSearchCV
from sklearn.metrics import accuracy_score, roc_curve, auc, roc_auc_scor
e
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from lightgbm import LGBMClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from scipy.sparse import hstack
```

In [0]:
```
# create a pipeline to convert the data into tfidf form
pipeline = Pipeline([
    ('bow', CountVectorizer(analyzer=remove_punc_stopwords)),  # strings
to token integer counts
    ('tfidf', TfidfTransformer())])
```

In [0]:
```
# Specify X and y
X = pipeline.fit_transform(df.tweet)
y = df.tag
final_X = pipeline.transform(df_test.tweet)


# Factorizing `y`
y = y.map({'Bad': 1, 'Good': 0})

# Train test split
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.1,
                                                    random_state=1)
```

In [0]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(3060, 8465)
(3060,)
(340, 8465)
(340,)
```

In [0]:
```
def gridSearchCV(model, params):
    """
    @param    model: sklearn estimator
    @param    params (dict): Dictionary of possible parameters

    @return   cv_results (DataFrame)
    """
    model_cv = GridSearchCV(model, param_grid=params, scoring='roc_auc', cv=5)
    model_cv.fit(X_train, y_train)
    cv_results = pd.DataFrame(model_cv.cv_results_)[['params', 'mean_test_score']].sort_values(['mean_test_score'],ascending=False)

    return cv_results
```

In [0]:
```
def evaluate(model, plotROC=False):
    """
    1. Plot ROC AUC of the test set
    2. Return the best threshold
    """
    model.fit(X_train, y_train)
    probs = model.predict_proba(X_test)
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(y_test, preds)
    roc_auc = auc(fpr, tpr)
    print(f'AUC: {roc_auc:.4f}')

    # Find optimal threshold
    rocDf = pd.DataFrame({'fpr': fpr, 'tpr':tpr, 'threshold':threshold})
    rocDf['tpr - fpr'] = rocDf.tpr - rocDf.fpr
    optimalThreshold = rocDf.threshold[rocDf['tpr - fpr'].idxmax()]
    print(optimalThreshold)

    # Get accuracy over the test set
    y_pred = np.where(preds >= optimalThreshold, 1, 0)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Accuracy: {accuracy*100:.2f}%')

    # Plot ROC AUC
    if plotROC:
        plt.title('Receiver Operating Characteristic')
        plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
        plt.legend(loc = 'lower right')
        plt.plot([0, 1], [0, 1],'r--')
        plt.xlim([0, 1])
        plt.ylim([0, 1])
        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.show()
```

4.2 Model Choosing

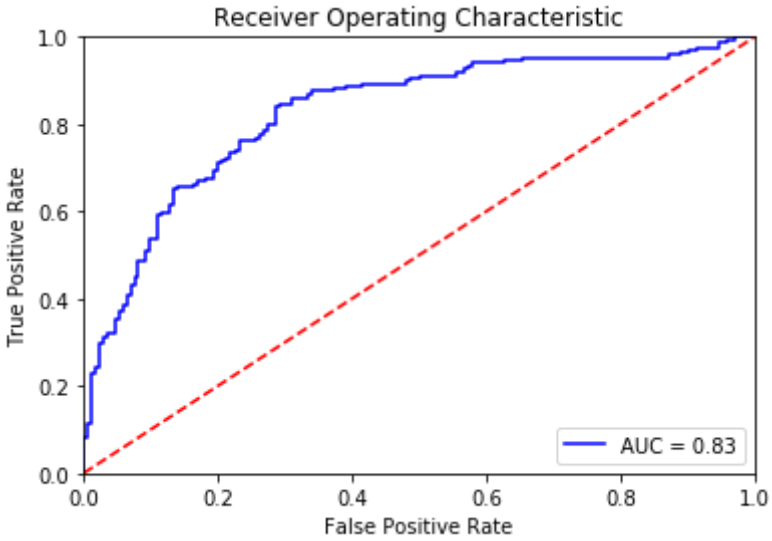
4.2.1 Logstic Regression

In [0]:
```
params = {'C': [0.0001,0.001,0.01, 0.1, 1, 10],
          'penalty':['l1','l2']}
logit = LogisticRegression()
print(gridSearchCV(logit, params))
```

	params	mean_test_score
9	{'C': 1, 'penalty': 'l2'}	0.824536
11	{'C': 10, 'penalty': 'l2'}	0.820655
7	{'C': 0.1, 'penalty': 'l2'}	0.806587
10	{'C': 10, 'penalty': 'l1'}	0.806196
8	{'C': 1, 'penalty': 'l1'}	0.801204
5	{'C': 0.01, 'penalty': 'l2'}	0.799022
3	{'C': 0.001, 'penalty': 'l2'}	0.798071
1	{'C': 0.0001, 'penalty': 'l2'}	0.797947
6	{'C': 0.1, 'penalty': 'l1'}	0.528724
0	{'C': 0.0001, 'penalty': 'l1'}	0.500000
2	{'C': 0.001, 'penalty': 'l1'}	0.500000
4	{'C': 0.01, 'penalty': 'l1'}	0.500000

In [0]:
```
logit = LogisticRegression(C=1,penalty='l2')
evaluate(logit, plotROC=True)
```

```
AUC: 0.8274
0.4657301612561441
Accuracy: 77.65%
```



4.2.2 Random Forest

In [0]:
```
params1 = {'bootstrap': [True, False]}
params2 = {'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, None]}
params3 = {'max_features': ['auto', 'sqrt']}
params4 = {'min_samples_leaf': [1, 2, 4]}
params5 = {'min_samples_split': [2, 5, 10]}
params6 = {'n_estimators': [100,200,400]}
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1)
print(gridSearchCV(rfc, params1))
```

```
                 params  mean_test_score
1  {'bootstrap': False}         0.778527
0   {'bootstrap': True}         0.755403
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False)
print(gridSearchCV(rfc, params2))
```

```
                  params  mean_test_score
5     {'max_depth': 60}         0.778939
8     {'max_depth': 90}         0.778706
10  {'max_depth': None}         0.778527
6     {'max_depth': 70}         0.777752
7     {'max_depth': 80}         0.777143
9    {'max_depth': 100}         0.775713
3     {'max_depth': 40}         0.770867
4     {'max_depth': 50}         0.770728
2     {'max_depth': 30}         0.762797
1     {'max_depth': 20}         0.749314
0     {'max_depth': 10}         0.724400
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=60
)
print(gridSearchCV(rfc, params3))
```

```
                    params  mean_test_score
0  {'max_features': 'auto'}         0.778939
1  {'max_features': 'sqrt'}         0.778939
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=60
)
print(gridSearchCV(rfc, params4))
```

```
                    params  mean_test_score
2  {'min_samples_leaf': 4}         0.788282
1  {'min_samples_leaf': 2}         0.783812
0  {'min_samples_leaf': 1}         0.778939
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=60
,min_samples_leaf=4)
print(gridSearchCV(rfc, params5))
```

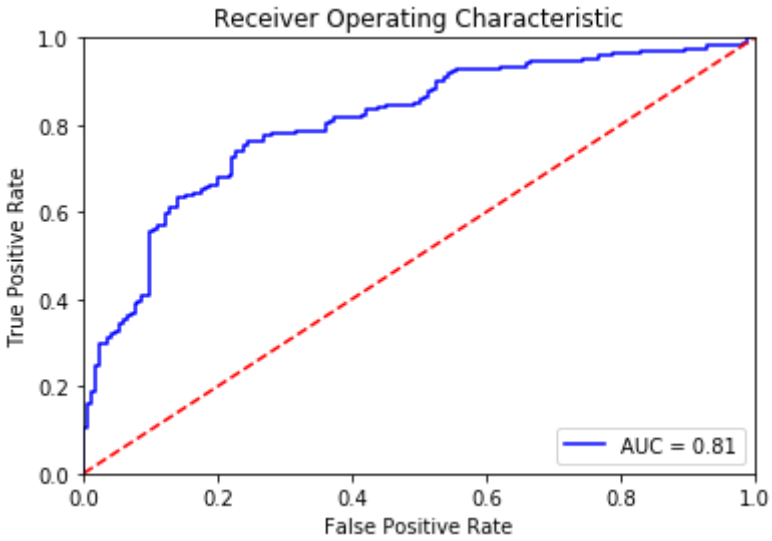
```
                       params  mean_test_score
2  {'min_samples_split': 10}         0.790325
0   {'min_samples_split': 2}         0.788282
1   {'min_samples_split': 5}         0.788282
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=60
,min_samples_leaf=4,min_samples_split=10)
print(gridSearchCV(rfc, params6))
```

```
                    params  mean_test_score
2  {'n_estimators': 400}         0.810360
1  {'n_estimators': 200}         0.810328
0  {'n_estimators': 100}         0.808333
```

In [0]:
```
rfc = RandomForestClassifier(random_state=1,bootstrap=False,max_depth=50
,min_samples_leaf=1,min_samples_split=2,n_estimators=400)
evaluate(rfc,plotROC=True)
```

```
AUC: 0.8062
0.49492399825567934
Accuracy: 75.88%
```



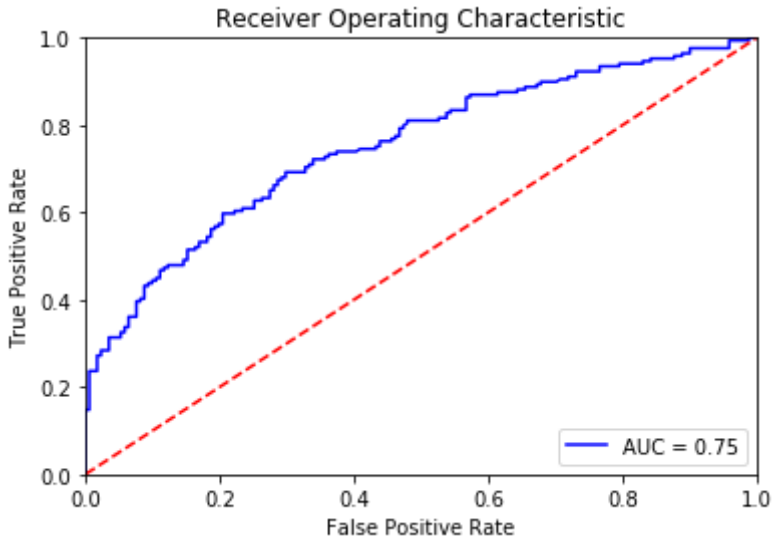
4.2.3 LightGBM

In [0]:
```
params = {'max_depth': [5, 6, 7]}
lightGBM = LGBMClassifier(learning_rate=0.05,
                          n_estimators=150,
                          num_leaves=8,
                          min_data_in_leaf=4,
                          max_depth=5,
                          max_bin=55,
                          bagging_fraction=0.78,
                          bagging_freq=5,
                          feature_fraction=0.24,
                          feature_fraction_seed=9,
                          bagging_seed=9,
                          min_sum_hessian_in_leaf=11)
print(gridSearchCV(lightGBM, params))
```

```
             params  mean_test_score
0  {'max_depth': 5}         0.735077
1  {'max_depth': 6}         0.734241
2  {'max_depth': 7}         0.732634
```

In [0]:
```
lightGBM = LGBMClassifier(learning_rate=0.05,
                          n_estimators=150,
                          num_leaves=8,
                          min_data_in_leaf=4,
                          max_depth=5,
                          max_bin=55,
                          bagging_fraction=0.78,
                          bagging_freq=5,
                          feature_fraction=0.24,
                          feature_fraction_seed=9,
                          bagging_seed=9,
                          min_sum_hessian_in_leaf=11,
                          random_state=1)
evaluate(lightGBM, plotROC=True)
```

```
AUC: 0.7517
0.49429624664042143
Accuracy: 69.71%
```



4.2.4 SVM

In [0]:
```
params1 = {'C': [0.1, 1, 3, 10],
           'kernel':['linear','rbf','poly']}
params2 = {'gamma':[0.1,1,10,100]}
params3 = {'degree':[0,1,2,3,4,5,6]}
```

In [0]:
```
svc = SVC()
print(gridSearchCV(svc, params1))
```

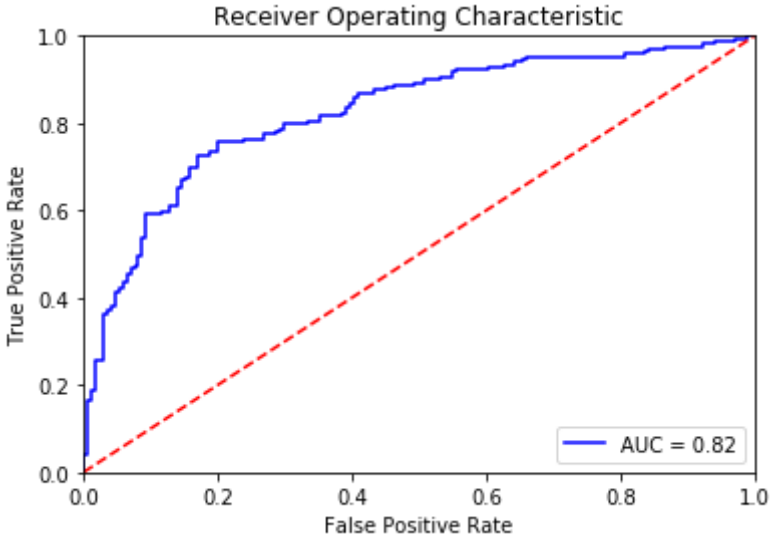
```
                            params  mean_test_score
3     {'C': 1, 'kernel': 'linear'}         0.820061
11     {'C': 10, 'kernel': 'poly'}         0.812522
5       {'C': 1, 'kernel': 'poly'}         0.812463
8       {'C': 3, 'kernel': 'poly'}         0.812390
2     {'C': 0.1, 'kernel': 'poly'}         0.812389
0   {'C': 0.1, 'kernel': 'linear'}         0.802255
6     {'C': 3, 'kernel': 'linear'}         0.800084
4        {'C': 1, 'kernel': 'rbf'}         0.798368
7        {'C': 3, 'kernel': 'rbf'}         0.798368
10      {'C': 10, 'kernel': 'rbf'}         0.798325
1      {'C': 0.1, 'kernel': 'rbf'}         0.798065
9    {'C': 10, 'kernel': 'linear'}         0.786141
```

In [0]:
```
svc = SVC(C=1,kernel='linear')
print(gridSearchCV(svc, params3))
```

```
          params  mean_test_score
0  {'degree': 0}         0.820061
1  {'degree': 1}         0.820061
2  {'degree': 2}         0.820061
3  {'degree': 3}         0.820061
4  {'degree': 4}         0.820061
5  {'degree': 5}         0.820061
6  {'degree': 6}         0.820061
```

In [0]:
```
svc = SVC(C=1,kernel='linear',probability=True,random_state=1)
evaluate(svc,plotROC=True)
```

```
AUC: 0.8249
0.49097749549852054
Accuracy: 77.94%
```



4.2.5 Naive Bays

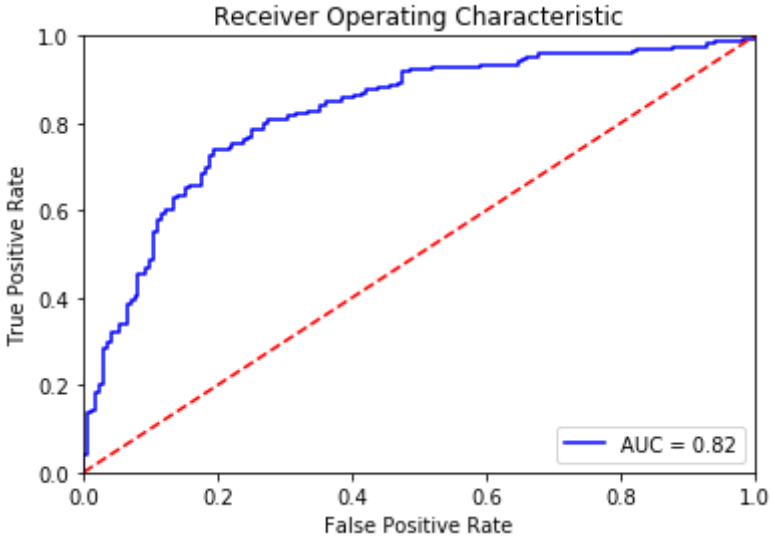
In [0]:
```
nab = MultinomialNB()
params1 = {'alpha': np.linspace(0.5, 1.5, 6), 'fit_prior': [True, False
]}
```

In [0]:
```
print(gridSearchCV(nab, params1))
```

```
                                  params  mean_test_score
10   {'alpha': 1.5, 'fit_prior': True}         0.831413
11  {'alpha': 1.5, 'fit_prior': False}         0.831413
8    {'alpha': 1.3, 'fit_prior': True}         0.830882
9   {'alpha': 1.3, 'fit_prior': False}         0.830882
6    {'alpha': 1.1, 'fit_prior': True}         0.830096
7   {'alpha': 1.1, 'fit_prior': False}         0.830096
4    {'alpha': 0.9, 'fit_prior': True}         0.828758
5   {'alpha': 0.9, 'fit_prior': False}         0.828758
2    {'alpha': 0.7, 'fit_prior': True}         0.826913
3   {'alpha': 0.7, 'fit_prior': False}         0.826913
0    {'alpha': 0.5, 'fit_prior': True}         0.823126
1   {'alpha': 0.5, 'fit_prior': False}         0.823126
```

In [0]:
```
nab = MultinomialNB(alpha=1.5,fit_prior=True)
evaluate(nab,plotROC=True)
```

```
AUC: 0.8226
0.558474328750573
Accuracy: 77.35%
```



5. Prediction Result

In [0]:
```
final_model = MultinomialNB(alpha=1.5,fit_prior=True)
final_model.fit(X,y)
```

Out[0]:
```
MultinomialNB(alpha=1.5, class_prior=None, fit_prior=True)
```

In [0]:
```
predictions = final_model.predict_proba(final_X)[:,1]
y_pred = np.where(predictions >= 0.38, 1, 0)
```

In [0]:
```
output = pd.DataFrame({
    'id':df_test.id,
    'tweet':df_test.tweet,
    'pred':y_pred
})
```

In [0]:
```
output.groupby('pred').count().drop('id',axis=1)
```

Out[0]:

pred	tweet
0	254
1	4301

In [0]:
```
Ruiling_Shen = output[output['pred']==0]
```

In [0]:
```
Ruiling_Shen.drop('pred',axis=1)
```

Out[0]:

	id	tweet
41	1290	Can't wait for @SouthwestAir's #NoLimits Inter...
43	1400	@VirginAmerica I always thought the girl with ...
49	1575	@Gogo @LinkedIn @VirginAmerica oh great so now...
70	2519	Thank you Carrie at @JetBlue Pittsburg for get...
82	2934	Knowing that your other flight left on time go...
...	...	...
4371	166542	@AlaskaAir 10 days til you take us to Maui. ...
4398	167320	@iaaronmitch @AmtrakNEC ehhhh I can never aff...
4423	168475	If I never flew @united again, I wouldn't be m...
4467	170440	Some partners in the "waiting on the summer th...
4543	173337	Jeff Smisek <<<------- CEO doesn'...

254 rows × 2 columns

In [0]:
```
Ruiling_Shen.to_csv(r'/content/drive/My Drive/SIMON /Fall B /CIS434 Soci
al Media /Final Project /data/Ruiling_Shen.csv',sep=',')
```