1. Background, Data Preparation, and Data Cleaning

Group: Group 11 - Alex Fung, Patrick Osborne
Dataset: Twitter US Airline Sentiment
Dataset link: https://www.kaggle.com/crowdflower/twitter-airline-sentiment (https://www.kaggle.com/crowdflower/twitter-airline-sentiment)

Background

For our course project we have chosen to conduct a sentiment analysis on a data set containing approximately 14.5 thousand tweets pertaining to 6 major US airlines. Going into this project we knew that we were interested in choosing a data set and problem that closely aligned to solving a topical business problem. We also wanted to pick a data set that would offer a certain degree of challenge and learning opportunities.

Our chosen data set aligns well with these goals for several reasons. Firstly, in the dozen or so years that Twitter has existed it has contributed to a significant shift in the way that companies interact with their customers, primarily from a customer service point of view, but additionally in terms of PR, marketing and even logistics. One tweet from the right (or wrong) person can set off a landslide of responses that can quickly become out of control. This is a particular concern for the US Airline industry. Over the past several years there have been several high-profile incidents causing negative public sentiment towards US airlines. We believe that a robust sentiment analysis model – specifically focused on identifying these negative sentiments before they become a larger problem - could help airlines better manage their PR crisis response and customer service. A sentiment analysis model tuned to identify negative tweets with a high degree of accuracy could help airlines analyse and learn from past Twitter trends and to rapidly identify new ones as they are occurring.

Secondly, from the perspective of data science students a Twitter data offers an interesting challenge in terms of cleaning, interpretation and prediction. As Twitter caps each message at a short character limit, complete English is rarely used. The data set is full of abbreviations, slang, emojis and Twitter functions such as hashtags, mentions and re-tweets. Cleaning these out while retaining the information in the original tweet will be important to developing an effective model. As with many customer service data sets, this one is also likely to be unbalanced towards the negative sentiment side. Though this aligns well with our goal to primarily identify negative sentiment tweets, this will be important to consider as we clean the data.

Data Preparation

The dataset downloaded manually from Kaggle contained a CSV and a Sqlite database file. Both files contain the same dataset, albeit in a different format. We chose to load the data using the CSV file because the dataset was not large enough to cause issues loading issues with the Pandas library. Also, the dataset had to be loaded with `UTF-8` encoding because we wanted to retain the information from emojis, which would be lost if we used another encoding, such as `CP-1252` We checked for nulls in `airline_sentiment` , `text` , and `negativereason` columns, although the dataset provided had no issues with nulls.

Below are links to the following Data Preparation Steps:

1. Load the data
2. Check for nulls

Load the data

In [1]:
```
import pandas as pd
pd.options.display.max_colwidth = None
```

It is important that we read the Tweets.csv with 'utf-8' encoding, so that we can extract the emojis properly

In [2]:
```
df_tweets = pd.read_csv("../data/Tweets.csv", encoding='utf-8')
df_tweets.head()
```

Out[2]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

The shape of the dataframe shows 14640 rows/observations, and 15 columns.

In [3]:
```
df_tweets.shape
```

Out[3]:
```
(14640, 15)
```

Check for Nulls

There are no nulls amongst the columns airline_sentiment and text, which is good

In [4]:
```
df_tweets['airline_sentiment'].isnull().sum()
```

Out[4]:
```
0
```

In [5]:
```
df_tweets['text'].isnull().sum()
```

Out[5]:
```
0
```

Column 'negativereason' has a few missing nulls, but they are null only if 'airline_sentiment' is positive or neutral.

In [6]:
```
df_tweets['negativereason'].isnull().sum()
```

Out[6]:
```
5462
```

In [7]:
```
df_tweets['airline_sentiment'].loc[df_tweets['negativereason'].isnull()].unique()
```

Out[7]:
```
array(['neutral', 'positive'], dtype=object)
```

Cleaning Data

Twitter data is notoriously "unclean" compared to the data of other NLP applications, such as newspaper articles, reviews, etc. As mentioned in the Background, Twitter's strict and small character limit means users must figure out ways of shortening their tweets into concise sentences. In practice, this means other forms of unorthodox written communication, such as Internet slang abbreviations, emojis, emoticons, hashtags, retweets, mentions, are employed regularly by users.

For some of these methods of communication, such as mentions and retweets, they provide zero or negative value to our sentiment analysis; as a result, it is in our best interest to remove such text. Mentions are used by users to direct their messages to handles of other other users, such as airline companies in our case. In this dataset, mentions almost always contained the handles of airline companies, which is not very useful if most of the tweets, regardless of positive, neutral, or negative sentiment, contain those mentions. Similarly, retweets were often the tweets of the PR Twitter accounts/handles of airline companies. Because the retweets do not reflect the true sentiment of the user's specific tweet, and the retweets themselves are often of neutral or positive sentiment (never negative, because the tweet would be deleted and the person in charge of the PR Twitter account/handle would be in serious trouble), it is logical to remove retweets from the dataset since retweets will only prove to be "noisy" to our sentiment analysis model.

On the other hand, other methods of communication such as emojis, emoticons, and hashtags could provide positive value to our sentiment analysis, since in theory visual/text elements such as emojis, and emoticons express a variety of emotions in a visual, or pseudo-visual image, which are possibly interlinked with the sentiment of the text. Hashtags could also be useful certain hashtags are associated with topics of positive, or negative sentimentality; for example, hashtags such as `#IceBucketChallenge` are linked with positive sentiments as the hashtag was used to encourage other people to perform acts of charity in a fun, positive manner.

Lastly, the text of the users will have to be cleaned to ensure the text can be properly tokenized by the Spacy model. Such actions of cleaning the text include removing HTML encoding and HTTP links, converting the text to lower case, translating Internet slang abbreviations to their full English expressions, removing stop words and numbers and punctuation, and performing the lemmatization of the remaining text.

Below are links to the following Data Preparation Steps:

1. Remove not useful columns
2. Create new text_cleaned column
3. Remove HTML encoding
4. Remove retweets
5. Remove mentions
6. Remove HTTP links
7. Extract Emojis and Emoticons
8. Extract and Remove Hashtags
9. Convert text to Lowercase
10. Internet Slang abbreviations
11. Removing StopWords and Punctuation and numbers, Lemmatization, and Tokenization

1. Remove not useful columns

Looking at the dataframe, we can see some columns will likely not be useful for our purposes of sentiment analysis, such timezones, number of retweets, etc. Therefore, we select the columns that we want, or think will be useful for our data exploration and analysis later.

In [8]:
```
df_tweets = df_tweets[
    ['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence', 'negativ
ereason', 'negativereason_confidence', 'airline', 'text']
]
df_tweets.head()
```

Out[8]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

2.Create new text_cleaned column

We create a column 'text_cleaned' that will contain the cleaned up version of 'text' column

In [9]:
```
df_tweets['text_cleaned'] = df_tweets['text']
```

3. Remove HTML encoding

The text has not been cleaned, as there is some HTML encoding left in the text, such as `&` . We will use BeautifulSoup and `lxml` package to remove the HTML encoding from the text.

Sanity Check

In [10]:
```
from bs4 import BeautifulSoup

#Example of what BeautifulSoup with lxml package does
#you may need to install lxml by 'pip install lxml' for this to work, then res
tart kernel
example1 = BeautifulSoup('Disappointed,UNITED did NOT feed small CHILDREN on a
5 &amp; half hour flight', 'lxml')
print(example1.get_text())
```

```
Disappointed,UNITED did NOT feed small CHILDREN on a 5 & half hour flight
```

Remove HTML Encoding from text

In [11]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].apply(lambda text: Beaut
ifulSoup(text, 'lxml').get_text())
df_tweets['text_cleaned']
```

Out[11]:
```
0
@VirginAmerica What @dhepburn said.
1
@VirginAmerica plus you've added commercials to the experience... tacky.
2
@VirginAmerica I didn't today... Must mean I need to take another trip!
3                                   @VirginAmerica it's really aggressive to
blast obnoxious "entertainment" in your guests' faces & they have little reco
urse
4
@VirginAmerica and it's a really big bad thing about it

...
14635
@AmericanAir thank you we got on a different flight to Chicago.
14636    @AmericanAir leaving over 20 minutes Late Flight. No warnings or com
munication until we were 15 minutes Late Flight. That's called shitty custome
r svc
14637
@AmericanAir Please bring American Airlines to #BlackBerry10
14638                   @AmericanAir you have my money, you change my flight,
and don't answer your phones! Any other suggestions so I can make my commitme
nt??
14639               @AmericanAir we have 8 ppl so we need 2 know how many se
ats are on the next flight. Plz put us on standby for 4 people on the next fl
ight?
Name: text_cleaned, Length: 14640, dtype: object
```

4. Remove retweets

Retweets are denoted in 'text' column as 'RT @another_user another_user's tweet'. We should remove retweets because we need to analyze the tweets of the users, not the retweets. In this situation, retweets provide no real value to our text exploration analysis as normally the users retweet to the airlines. Retweets are based off the tweets from the specified airlines' Twitter PR, and therefore, are likely going to be either of neutral or positive sentiment. Removing such retweets will hopefully remove any noise that will prevent the models from classifying sentiment.

Sanity Check

In [12]:
```
regex_to_replace= r'RT \@.*'
replace_value= ''
```

In [13]:
```
import re
example1 = 'Awesome! RT @VirginAmerica: Watch nominated films at 35,000 feet.
 #MeetTheFleet #Oscars http://t.co/DnStITRzWy'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[13]:
```
'Awesome! '
```

Remove retweets from text

In [14]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[14]:
```
0
@VirginAmerica What @dhepburn said.
1
@VirginAmerica plus you've added commercials to the experience... tacky.
2
@VirginAmerica I didn't today... Must mean I need to take another trip!
3                                   @VirginAmerica it's really aggressive to
blast obnoxious "entertainment" in your guests' faces & they have little reco
urse
4
@VirginAmerica and it's a really big bad thing about it

...
14635
@AmericanAir thank you we got on a different flight to Chicago.
14636    @AmericanAir leaving over 20 minutes Late Flight. No warnings or com
munication until we were 15 minutes Late Flight. That's called shitty custome
r svc
14637
@AmericanAir Please bring American Airlines to #BlackBerry10
14638                   @AmericanAir you have my money, you change my flight,
and don't answer your phones! Any other suggestions so I can make my commitme
nt??
14639                @AmericanAir we have 8 ppl so we need 2 know how many se
ats are on the next flight. Plz put us on standby for 4 people on the next fl
ight?
Name: text_cleaned, Length: 14640, dtype: object
```

5. Remove mentions

Sometimes, users use mentions (for example, tweet mentions include @VirginAirlines, @JetBlue, etc., in other words, the airlines' handle). They normally appear in the beginning of the users' tweets. These mentions are not useful for sentiment analysis purposes because the vast majority of tweets have some sort of mention to one of the 6 airline companies.

Sanity Check

In [15]:
```
regex_to_replace = r'\@[\w\d]*'
replace_value= ''
```

In [16]:
```
import re
example1 = '@VirginAmerica Thank you for the follow'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[16]:
```
' Thank you for the follow'
```

Remove mentions from text

In [17]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[17]:
```
0
What  said.
1
plus you've added commercials to the experience... tacky.
2
I didn't today... Must mean I need to take another trip!
3                                          it's really aggressive to blast obnox
ious "entertainment" in your guests' faces & they have little recourse
4
and it's a really big bad thing about it

...
14635
thank you we got on a different flight to Chicago.
14636     leaving over 20 minutes Late Flight. No warnings or communication u
ntil we were 15 minutes Late Flight. That's called shitty customer svc
14637
Please bring American Airlines to #BlackBerry10
14638                     you have my money, you change my flight, and don't a
nswer your phones! Any other suggestions so I can make my commitment??
14639                  we have 8 ppl so we need 2 know how many seats are on t
he next flight. Plz put us on standby for 4 people on the next flight?
Name: text_cleaned, Length: 14640, dtype: object
```

6. Remove HTTP links

Users attach http links occasionally in their tweets. We need to remove HTTP links from the text, since they provide no real value to our sentiment analysis as well. From what we can see in the data, they are mostly links to articles.

Sanity Check

In [18]:
```
regex_to_replace = r'https*://[^\s]*'
replace_value = ''
```

In [19]:
```
import re
example1 = '@VirginAmerica when are you putting some great deals from PDX to L
AS or from LAS to PDX show me your love! http://t.co/enIQg0buzj'
example1 = re.sub(regex_to_replace, replace_value, example1)
example1
```

Out[19]:
```
'@VirginAmerica when are you putting some great deals from PDX to LAS or from
LAS to PDX show me your love! '
```

Removing HTTP Links from text

In [20]:
```
df_tweets['text_cleaned'] = df_tweets['text_cleaned'].replace(to_replace=regex
_to_replace, value=replace_value, regex=True)
df_tweets['text_cleaned']
```

Out[20]:
```
0
What  said.
1
plus you've added commercials to the experience... tacky.
2
I didn't today... Must mean I need to take another trip!
3                                          it's really aggressive to blast obnox
ious "entertainment" in your guests' faces & they have little recourse
4
and it's a really big bad thing about it

...
14635
thank you we got on a different flight to Chicago.
14636     leaving over 20 minutes Late Flight. No warnings or communication u
ntil we were 15 minutes Late Flight. That's called shitty customer svc
14637
Please bring American Airlines to #BlackBerry10
14638                     you have my money, you change my flight, and don't a
nswer your phones! Any other suggestions so I can make my commitment??
14639                 we have 8 ppl so we need 2 know how many seats are on t
he next flight. Plz put us on standby for 4 people on the next flight?
Name: text_cleaned, Length: 14640, dtype: object
```

7. Extract Emojis and Emoticons

Rather than removing emojis and emoticons, emojis and emoticons can be seen as an integral part of the Internet language. Therefore, we should extract emojis and emoticons from the text if they exist, as they may be good features for sentiment analysis for "Internet"-speak.

Emojis are special characters which are shown as actual visual images, whereas emoticons are keyboard characters arranged in a certain format so that it represents a human-like facial expression. Emoticons are not as commonly used compared to emojis anymore, but are still used occassionally by people, and it is in our best interests to also extract emoticons as well.

We will use a third-party Python library called 'emot', which provides the ability to recognize and extract both emojis and emoticons. Github can be found here: https://github.com/NeelShah18/emot (https://github.com/NeelShah18/emot). There was an issue with the library where emot.emoticons would return different JSON structures, either `{'flag': False}` or `{'value': [], 'mean': [], 'location': [], 'flag': False}` whenever emoticons were not found. Upon manually investigating a few examples with such issues, we found out that although this was an issue, it was true that there were no emoticons in the texts of those examples. Therefore, we got around this issue by writing a variety of try/except catches.

Testing the `emot` package manually

In [21]:
```
#Sanity check
import emot
text = "I love python 👦 :-)"
print(emot.emoji(text))
print(emot.emoticons(text))
```

```
{'value': ['👦'], 'mean': [':man:'], 'location': [[14, 14]], 'flag': True}
{'value': [':-)'], 'location': [[16, 19]], 'mean': ['Happy face smiley'], 'fl
ag': True}
```

In [22]:
```
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 569198104806699008]
print(df_example1['text_cleaned'].to_string(index=False))
print(emot.emoji(df_example1['text_cleaned'].to_string(index=False)))
print(emot.emoticons(df_example1['text_cleaned'].to_string(index=False)))
```

```
  hahaha 😂 YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!💗
{'value': ['😂', '💗'], 'mean': [':face_with_tears_of_joy:', ':growing_hear
t:'], 'location': [[9, 9], [51, 51]], 'flag': True}
{'value': [], 'location': [], 'mean': [], 'flag': False}
```

In [23]:
```
df_example2 = df_tweets.loc[df_tweets['tweet_id'] == 568890074164809728]
print(df_example2['text_cleaned'].to_string(index=False))
print(emot.emoji(df_example2['text_cleaned'].to_string(index=False)))
print(emot.emoticons(df_example2['text_cleaned'].to_string(index=False)))
```

```
  we have a hot female pilot! Sweet! DCA to SFO! :-)
{'value': [], 'mean': [], 'location': [], 'flag': False}
{'value': [':-)'], 'location': [[49, 52]], 'mean': ['Happy face smiley'], 'fl
ag': True}
```

Extracing Emojis and Emoticons from text

In [24]:
```
#this function will remove any emojis
#accepts the following parameters:
#(1) text_cleaned: roughly cleaned text in String to parse through and remove
 emojis
#(2) emojis_flag: boolean created by emot.emoji(...), can be accessed by calli
ng 'flag' key in dictionary (for more information see above)
#(3) emojis: emojis object created by emot.emoji(...), can be accessed by call
ing 'value' key in dictionary (for more information see above)

#returns the cleaned up text without emojis

def remove_emojis(text_cleaned, emojis_flag, emojis):
    text_cleaned_no_emojis = text_cleaned

    #print('text_cleaned @ remove_emojis: ' + text_cleaned)
    #print('emojis_flag: ' + str(emojis_flag))
    #print('emojis: ' + str(emojis))


    #if flag is True, that means there are emojis, and we need to remove them
    if emojis_flag:
        for i in emojis:

            #rather than use location, we will match by String and see if we c
an remove it, because I'm lazy af lol
            #print(str(i))
            text_cleaned_no_emojis = text_cleaned_no_emojis.replace(i, '')

        if text_cleaned_no_emojis == text_cleaned:
            print('Uh...Houston, we have a problem...for the following text: '
+ text_cleaned + ', for row: ' + str(i))
            print('The following emojis were not removed: ' + str(emojis))

    #print('resulting text_cleaned_no_emojis: ' + text_cleaned_no_emojis)
    return text_cleaned_no_emojis
```

In [25]:
```
#this function will remove any emoticons
#accepts the following parameters:
#(1) text_cleaned: roughly cleaned text in String to parse through and remove
 emoticons
#(2) emoticons_flag: boolean created by emot.emoticons(...), can be accessed b
y calling 'flag' key in dictionary (for more information see above)
#(3) emoticons: emojis object created by emot.emoticons(...), can be accessed
 by calling 'value' key in dictionary (for more information see above)

#returns the cleaned up text without emoticons

def remove_emoticons(text_cleaned, emoticons_flag, emoticons):
    text_cleaned_no_emoticons = text_cleaned

    #print('text_cleaned @ remove_emoticons: ' + text_cleaned)
    #print('emoticons_flag: ' + str(emoticons_flag))
    #print('emoticons: ' + str(emoticons))

    #if flag is True, that means there are emoticons, and we need to remove th
em
    if emoticons_flag:
        for i in emoticons:

            #rather than use location, we will match by String and see if we c
an remove it, because I'm lazy af lol
            #print(str(i))
            text_cleaned_no_emoticons = text_cleaned_no_emoticons.replace(i,
'')

        if text_cleaned_no_emoticons == text_cleaned:
            print('Uh...Houston, we have a problem...for the following text: '
+ text_cleaned + ', for row: ' + str(i))
            print('The following emojis were not removed: ' + str(emoticons))

    #print('resulting text_cleaned_no_emoticons: ' + text_cleaned_no_emoticon
s)
    return text_cleaned_no_emoticons
```

In [26]:
```
#this function will extract any emojis into a separate 'emojis' column,
#while also removing said emojis from column: 'text_cleaned'
#to form a new column: 'text_cleaned_without_emojis_emoticons'

#returns dataframe with the aforementioned new columns

import emot

#set logging so we don't output the try/except log messages unless if we want
 them
#default level if WARNING
import logging

#in 'emojis_emoticons' column, it will hold the emot.emoji return dictionary
 (see above for examples)
def extract_emojis(df_tweets, ):
    df_tweets_2 = df_tweets
    df_tweets_2['emojis_flag'] = ''
    df_tweets_2['emojis'] = ''
    df_tweets_2['emoticons_flag'] = ''
    df_tweets_2['emoticons'] = ''

    #easier to just write out code to loop through dataframe
    for i, row in df_tweets_2.iterrows():
        #print(i)
        #print(row)
        text_cleaned = df_tweets_2.at[i, 'text_cleaned']
        emojis = emot.emoji(text_cleaned)
        emoticons = emot.emoticons(text_cleaned)
        #print('EMOJIS: ' + str(emojis))
        #print('EMOTICONS: ' + str(emoticons))

        ##When using emot.emoticons, the output when flag=False is inconsisten
t, it either is
        #(a) {'value': [], 'mean': [], 'location': [], 'flag': False}, or
        #(b) {'flag': False}

        #Therefore we have a bunch of try-except the aforementioned exception
 that will be raised, and set the values manually
        try:
            df_tweets_2.at[i, 'emojis_flag'] = emojis['flag']
        except:
            logging.debug('Unable to grab emojis flag at row number: ' + str(i
))
            df_tweets_2.at[i, 'emojis_flag'] = False

        try:
             df_tweets_2.at[i, 'emojis'] = emojis['value']
        except:
            logging.debug('Unable to grab emojis value at row number: ' + str(
i))
            df_tweets_2.at[i, 'emojis'] = []

        try:
            df_tweets_2.at[i, 'emoticons_flag'] = emoticons['flag']
        except:
```

```
            logging.debug('Unable to grab emoticons flag at row number: ' + st
r(i))
            df_tweets_2.at[i, 'emoticons_flag'] = False


        try:
            df_tweets_2.at[i, 'emoticons'] = emoticons['value']
        except:
            logging.debug('Unable to grab emoticons value at row number: ' + s
tr(i))
            df_tweets_2.at[i, 'emoticons'] = []

        #afterwards, we need to remove emojis and emoticons from the
        df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'] = remove_em
ojis(
            text_cleaned,
            df_tweets_2.at[i, 'emojis_flag'],
            df_tweets_2.at[i, 'emojis']
        )

        df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'] = remove_em
oticons(
            df_tweets_2.at[i, 'text_cleaned_without_emojis_emoticons'],
            df_tweets_2.at[i, 'emoticons_flag'],
            df_tweets_2.at[i, 'emoticons']
        )

    return df_tweets_2
```

In [27]:
```
#df_tweets['text_cleaned'] = df_tweets['text_cleaned'].apply(lambda text: Beau
tifulSoup(text, 'lxml').get_text())
#df_tweets['text_cleaned']
df_tweets = extract_emojis(df_tweets)
#df_tweets.head()
```

Sanity check

In [28]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 569198104806699008]
print(df_example1['emojis_flag'])
print(df_example1['emojis'])
print(df_example1['text'])
print(df_example1['text_cleaned'])
print(df_example1['text_cleaned_without_emojis_emoticons'])
```

```
238    True
Name: emojis_flag, dtype: object
238    [😂, 💙]
Name: emojis, dtype: object
238    @VirginAmerica hahaha 😂@VirginAmerica YOU GUYS ARE AMAZING. I LOVE YO
U GUYS!!!💙
Name: text, dtype: object
238     hahaha 😂 YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!💙
Name: text_cleaned, dtype: object
238     hahaha  YOU GUYS ARE AMAZING. I LOVE YOU GUYS!!!
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

In [29]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example1 = df_tweets.loc[df_tweets['tweet_id'] == 570270684619923457]
print(df_example1['emojis_flag'])
print(df_example1['emojis'])
print(df_example1['text'])
print(df_example1['text_cleaned'])
print(df_example1['text_cleaned_without_emojis_emoticons'])
```

```
18    True
Name: emojis_flag, dtype: object
18    [❤, ☺, 👍]
Name: emojis, dtype: object
18    I ♡ flying @VirginAmerica. ☺👍
Name: text, dtype: object
18    I ♡ flying . ☺👍
Name: text_cleaned, dtype: object
18    I  flying .
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

In [30]:
```
#Sanity check: briefly check some examples where we know emojis and emoticons
 do exist
df_example2 = df_tweets.loc[df_tweets['tweet_id'] == 568890074164809728]
print(df_example2['emoticons_flag'])
print(df_example2['emoticons'])
print(df_example2['text'])
print(df_example2['text_cleaned'])
print(df_example2['text_cleaned_without_emojis_emoticons'])
```

```
277    True
Name: emoticons_flag, dtype: object
277    [:-)]
Name: emoticons, dtype: object
277    @VirginAmerica we have a hot female pilot! Sweet! DCA to SFO! :-)
Name: text, dtype: object
277     we have a hot female pilot! Sweet! DCA to SFO! :-)
Name: text_cleaned, dtype: object
277     we have a hot female pilot! Sweet! DCA to SFO!
Name: text_cleaned_without_emojis_emoticons, dtype: object
```

8. Extract and Remove Hashtags

Some of the tweets have hashtag, which are keyword phrases spelled out with no spaces, and a pound sign at the front. They could offer more insight into the sentiment of the tweet. There were 3669 hashtags in the dataset in 2489 rows (some tweets evidently had more than one hashtag).

For now, the easiest approach would be to remove the hashtags and the words inside during text cleaning. If we have time, we can try to analyze the best approaches to extract the words from the hashtags, and split them into meaningful words. This would be fairly challenging, however, as some tweets have multiple words that are not capitalized, so there is not an easy, fool-proof way of extracing hashtags into words.

We also decided to extract the hashtags into a list inside a separate column for further data exploration and analysis.

In [31]:
```
print(len(df_tweets.loc[df_tweets['text'].str.contains('#')]))
```

```
2489
```

In [32]:
```
#extracts hashtags into a list, which is put into a new column called 'hashtag
s', whilst also removing hashtags from the text_cleaned
def extract_and_remove_hashtags(df_tweets):
    regex_to_replace= r'#(\w+)'
    replace_value= ''
    df_tweets['hashtags'] = ''
    df_tweets['text_cleaned_without_emojis_emoticons_hashtags'] = ''

    for i, row in df_tweets.iterrows():
        df_tweets.at[i, 'hashtags'] = re.findall(regex_to_replace, df_tweets.a
t[i, 'text_cleaned'])
        df_tweets.at[i, 'text_cleaned_without_emojis_emoticons_hashtags'] = re
.sub(regex_to_replace, replace_value,  df_tweets.at[i, 'text_cleaned'])

        #create hashtags_flag
        if df_tweets.at[i, 'hashtags'] == []:
            df_tweets.at[i, 'hashtags_flag'] = False
        else:
            df_tweets.at[i, 'hashtags_flag'] = True

    return df_tweets
```

In [33]:
```
df_tweets = extract_and_remove_hashtags(df_tweets)
df_tweets.loc[df_tweets['tweet_id'] == 569587242672398336]
```

Out[33]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	nega
14637	569587242672398336	neutral	1.0	NaN	

9. Convert text to Lowercase

We need to convert the text to lower case. This is done so that when we tokenize the words, there won't be words that are grouped separately just because of case sensitivity.

In [34]:
```
df_tweets['text_cleaned_lower_case'] = \
    df_tweets['text_cleaned_without_emojis_emoticons_hashtags'].apply(lambda t
ext: text.lower())
df_tweets.head()
```

Out[34]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

10. Internet Slang abbreviations

We need to convert the Internet slang abbreviations into readable English lexicon. To do so, we will use a dictionary of commonly used slang words, which we have used partly from here: https://github.com/Deffro/text-preprocessing-techniques/blob/master/slang.txt (https://github.com/Deffro/text-preprocessing-techniques/blob/master/slang.txt). We modified the dictionary to be in CSV format, as well as changing some of the key-value pairs as they were improperly written. Each Internet slang abbreviation key is linked to its respective value, which would be the complete form of the abbreviation.

In [35]:
```
df_slang = pd.read_csv('..\data\slang.csv')
df_slang
```

Out[35]:

	slang_abbreviation	complete_form
0	2day	today
1	2nite	tonight
2	4u	for you
3	4ward	forward
4	a3	anyplace, anywhere, anytime
...	...	...
285	yuge	huge
286	yw	you are welcome
287	ywa	you are welcome anyway
288	zomg	oh my god!
289	zzz	sleeping

290 rows × 2 columns

In [36]:
```
def replace_abbreviation(text, abbreviation, complete_form):
    return re.sub(abbreviation, complete_form, text)
```

In [37]:
```
#Sanity check, testing above function
regex_expression = re.compile('\\b'+'plz'+'\\b')
regex_to_replace =  'please'

replace_abbreviation(
    'o we fly straight into sfo and honululu gets pushed back 3.5 hours and no
w it looks like more delays.  i beg of you plz sort this out soon!',
    regex_expression,
    regex_to_replace
)
```

Out[37]:
```
'o we fly straight into sfo and honululu gets pushed back 3.5 hours and now i
t looks like more delays.  i beg of you please sort this out soon!'
```

In [38]:
```
#Another sanity check, but a negative case this time around
#the word 'straight' should remain the same, even though the dictionary contai
ns the abbreviation 'aight'
regex_expression = re.compile('\\b'+'aight'+'\\b')
regex_to_replace = 'alright'

replace_abbreviation(
    'o we fly straight into sfo and honululu gets pushed back 3.5 hours and no
w it looks like more delays.  i beg of you plz sort this out soon!',
    regex_expression,
    regex_to_replace
)
```

Out[38]:
```
'o we fly straight into sfo and honululu gets pushed back 3.5 hours and now i
t looks like more delays.  i beg of you plz sort this out soon!'
```

In [39]:
```
#this function will find slang abbreviations and replace them with the complet
e forms
#needs df_tweets and df_slang
#returns a new dataframe with new column 'text_cleaned_without_emojis_emoticon
s_hashtags_abbreviations'

#we will iterate over every row/tweet and and see if there are instances of sl
ang abbreviations
#computationally expensive, but because there are only 14000 rows, it's not to
o bad
#there is no easy way of detecting whether a word is an abbreviation or not us
ing Spacy, NLTK, or spellchecker
def find_slang_abbreviations_and_replace_with_complete_form(df_tweets, df_slan
g):
    df_tweets['text_cleaned_no_abbreviations'] = ''

    for i, tweet_row in df_tweets.iterrows():
        df_tweets.at[i, 'text_cleaned_no_abbreviations'] = \
            df_tweets.at[i, 'text_cleaned_lower_case']

        if i % 1000 == 0:
            print('at row number: ' + str(i))

        for j, slang_row in df_slang.iterrows():
            #print(slang_row)
            slang_abbreviation = df_slang.at[j, 'slang_abbreviation']
            complete_form = df_slang.at[j, 'complete_form']

            #print("slang_abbreviation: " + slang_abbreviation)
            #print("complete_form: " + complete_form)

            regex_expression = re.compile('\\b'+slang_abbreviation+'\\b')
            regex_to_replace = complete_form

            df_tweets.at[i, 'text_cleaned_no_abbreviations'] = \
                replace_abbreviation(
                    df_tweets.at[i, 'text_cleaned_no_abbreviations'],
                    regex_expression,
                    regex_to_replace
            )
            #print("current: " + df_tweets.at[i, 'text_cleaned_no_abbreviation
s'])


    return df_tweets
```

In [40]:
```
#create new column 'text_cleaned_no_abbreviations' in df_tweets
#by calling find_slang_abbreviations_and_replace_with_complete_form function
df_tweets = find_slang_abbreviations_and_replace_with_complete_form(df_tweets,
df_slang)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [41]:
```
df_tweets.head()
```

Out[41]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

In [42]:
```
#sanity check
df_tweets.loc[df_tweets['tweet_id'] == 568899587424931840]
```

Out[42]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negat
2788	568899587424931840	negative	1.0	Late Flight	

11. Removing StopWords and Punctuation and numbers, Lemmatization, and Tokenization

We want to remove stop words because most common words such as "the", "and", "I", "you" are not likely to add any value to our sentiment analysis. Removing stopwords will therefore reduce the noise that will reduce the effectiveness of our eventual sentiment analysis model. Removing stopwords is also beneficial in the sense that it also removes common contractions such as "I've, you're, etc."

We also want to remove punctuation because punctuation is not useful to our sentiment analysis. Punctutation might help determine the intensity, or polarity of text, but that is not relevant to sentiment. Similar to punctuation, we also want to remove numbers.

Lastly, we will need to perform lemmatization, so that we can reduce the words to their root form

In [43]:
```
#load spacy model
import spacy

nlp = spacy.load('en_core_web_md')
```

In [44]:
```
df_tweets['text_list_no_stop_words'] = ''
df_tweets['lemmas_list'] = ''

for i, row in df_tweets.iterrows():
    if i % 1000 == 0:
        print('at row number: ' + str(i))

    text = df_tweets.at[i, 'text_cleaned_no_abbreviations']

    #tokenize text into list of tokens
    token_list = nlp(text)

    text_list_no_stop_words = []
    lemmas_list = []

    #remove stop words, and lemmatize
    for token in token_list:
        #print(str(token.is_stop))
        #print(str(token.pos_))

        #if token is not a stop word, and not punctuation, and not a number,
        #then it is useful to us and we store them in our lists
        if (token.is_stop == False) & (not token.is_punct) & (not token.is_spa
ce) & (not token.is_digit):
            text_list_no_stop_words.append(token.text)
            lemmas_list.append(token.lemma_)

    #print('text_list_no_stop_words:' + str(text_list_no_stop_words))
    #print('lemmas_list:' + str(lemmas_list))
    df_tweets.at[i, 'text_list_no_stop_words'] = " ".join(text_list_no_stop_wo
rds)
    df_tweets.at[i, 'lemmas_list'] = " ".join(lemmas_list)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [45]:
```
#sanity check
df_tweets.head()
```

Out[45]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativer
0	570306133677760513	neutral	1.0000	NaN	
1	570301130888122368	positive	0.3486	NaN	
2	570301083672813571	neutral	0.6837	NaN	
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	

Cleaning data is now complete, we'll save the dataframe to a csv.

In [46]:
```
df_tweets.to_csv('..\data\Tweets_cleaned.csv', index = False)
```

Data Exploration and Analysis

Reference

Some example code used (with modification) from https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a (https://medium.com/@datanizing/modern-text-mining-with-python-part-1-of-5-introduction-cleaning-and-linguistics-647f9ec85b6a)

Initial Setup and Imports

All packages used in this notebook are imported here and basic configuration options are set.

In [1]:
```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np

# adjust pandas display
pd.options.display.max_columns = 30
pd.options.display.max_rows = 100
pd.options.display.float_format = '{:.2f}'.format
pd.options.display.precision = 2
pd.options.display.max_colwidth = -1

from collections import Counter
from wordcloud import WordCloud, STOPWORDS
```

In [2]:
```
# Import matplotlib and seaborn and adjust some defaults
%matplotlib inline
%config InlineBackend.figure_format = 'svg'

from matplotlib import pyplot as plt
plt.rcParams['figure.dpi'] = 100

import seaborn as sns
sns.set_style("whitegrid")
```

Load Data into Pandas

This is the cleaned data file that was created in the previous, Data Preparation notebook.

In [3]:
```
df = pd.read_csv("C:\git\CSML1010-Group_11-Final-Project\proposal\Tweets_cleaned.csv")
```

Basic Properties of the Dataset

Here we examine the features in the data set, including the new ones created in the previous notebook.

In [4]:
```
# list column names and datatypes
df.dtypes
```

Out[4]:
```
tweet_id                                          int64
airline_sentiment                                 object
airline_sentiment_confidence                      float64
negativereason                                    object
negativereason_confidence                         float64
airline                                           object
text                                              object
text_cleaned                                      object
emojis_flag                                       bool
emojis                                            object
emoticons_flag                                    bool
emoticons                                         object
text_cleaned_without_emojis_emoticons             object
hashtags                                          object
text_cleaned_without_emojis_emoticons_hashtags    object
text_cleaned_lower_case                           object
text_cleaned_no_abbreviations                     object
text_list_no_stop_words                           object
lemmas_list                                       object
dtype: object
```

In [5]:
```
# select a sample of some data frame columns
df[['airline_sentiment', 'airline_sentiment_confidence', 'negativereason', 'ne
gativereason_confidence', 'airline', 'text']] \
  .sample(10, random_state=0)
```

Out[5]:

	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence
13983	negative	0.62	Late Flight	0.62
14484	negative	0.70	Bad Flight	0.36
6403	negative	1.00	Cancelled Flight	1.00
9653	negative	1.00	Can't Tell	1.00
13268	negative	1.00	Customer Service Issue	1.00
2384	neutral	1.00	NaN	nan
9613	negative	0.67	Customer Service Issue	0.34

	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence
11612	negative	0.71	Late Flight	0.71
9252	negative	1.00	Late Flight	1.00
13923	negative	1.00	Customer Service Issue	1.00

In [6]:
```
# length of a dataframe (# of rows/unique tweets)
len(df)
```

Out[6]: 14640

In [7]:
```
# number of values per column
df.count()
```

Out[7]:
```
tweet_id                                        14640
airline_sentiment                               14640
airline_sentiment_confidence                    14640
negativereason                                   9178
negativereason_confidence                       10522
airline                                         14640
text                                            14640
text_cleaned                                    14637
emojis_flag                                     14640
emojis                                          14640
emoticons_flag                                  14640
emoticons                                       14640
text_cleaned_without_emojis_emoticons           14636
hashtags                                        14640
text_cleaned_without_emojis_emoticons_hashtags  14637
text_cleaned_lower_case                         14637
text_cleaned_no_abbreviations                   14637
text_list_no_stop_words                         14561
lemmas_list                                     14561
dtype: int64
```

In [8]:
```
# size info, including memory consumption
df.info(memory_usage='deep')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 19 columns):
 #   Column                                        Non-Null Count  Dtype
---  ------                                        --------------  -----
 0   tweet_id                                      14640 non-null  int64
 1   airline_sentiment                             14640 non-null  object
 2   airline_sentiment_confidence                  14640 non-null  float64
 3   negativereason                                9178 non-null   object
 4   negativereason_confidence                     10522 non-null  float64
 5   airline                                       14640 non-null  object
 6   text                                          14640 non-null  object
 7   text_cleaned                                  14637 non-null  object
 8   emojis_flag                                   14640 non-null  bool
 9   emojis                                        14640 non-null  object
 10  emoticons_flag                                14640 non-null  bool
 11  emoticons                                     14640 non-null  object
 12  text_cleaned_without_emojis_emoticons         14636 non-null  object
 13  hashtags                                      14640 non-null  object
 14  text_cleaned_without_emojis_emoticons_hashtags 14637 non-null  object
 15  text_cleaned_lower_case                       14637 non-null  object
 16  text_cleaned_no_abbreviations                 14637 non-null  object
 17  text_list_no_stop_words                       14561 non-null  object
 18  lemmas_list                                   14561 non-null  object
dtypes: bool(2), float64(2), int64(1), object(14)
memory usage: 21.8 MB
```

Exploring Column Summaries

The pandas `describe` method computes statistical summaries for each of the columns of a dataframe. The results are different for categorical and numerical features.

Summary for Categorical Features

Based on an initial examination of the data set, we've identified the 'airline_sentiment', 'negativereason', 'airline' and 'text' columns as the interesting categorial features. The other columns contain data that is either too incomplete to use, or is not revelant to our stated problem.

In [9]:
```
columns = [col for col in df.columns]
columns
```

Out[9]:
```
['tweet_id',
 'airline_sentiment',
 'airline_sentiment_confidence',
 'negativereason',
 'negativereason_confidence',
 'airline',
 'text',
 'text_cleaned',
 'emojis_flag',
 'emojis',
 'emoticons_flag',
 'emoticons',
 'text_cleaned_without_emojis_emoticons',
 'hashtags',
 'text_cleaned_without_emojis_emoticons_hashtags',
 'text_cleaned_lower_case',
 'text_cleaned_no_abbreviations',
 'text_list_no_stop_words',
 'lemmas_list']
```

In [10]:
```
# describe categorical columns of type np.object
df[['airline_sentiment', 'negativereason', 'airline', 'text']] \
  .describe(include=np.object) \
  .transpose()
```

Out[10]:

	count	unique	top	freq
airline_sentiment	14640	3	negative	9178
negativereason	9178	10	Customer Service Issue	2910
airline	14640	6	United	3822
text	14640	14427	@united thanks	6

In [11]:
```
df['airline_sentiment'].value_counts()[:10]
```

Out[11]:
```
negative    9178
neutral     3099
positive    2363
Name: airline_sentiment, dtype: int64
```

In [12]:
```
df['negativereason'].value_counts()[:10]
```

Out[12]:
```
Customer Service Issue         2910
Late Flight                    1665
Can't Tell                     1190
Cancelled Flight               847
Lost Luggage                   724
Bad Flight                     580
Flight Booking Problems        529
Flight Attendant Complaints    481
longlines                      178
Damaged Luggage                74
Name: negativereason, dtype: int64
```

In [13]:
```
df['airline'].value_counts()[:10]
```

Out[13]:
```
United            3822
US Airways        2913
American          2759
Southwest         2420
Delta             2222
Virgin America    504
Name: airline, dtype: int64
```

In [14]:
```
df['text'].value_counts()[:10]
```

Out[14]:
```
@united thanks                                                  6
@SouthwestAir sent                                              5
@AmericanAir thanks                                             5
@JetBlue thanks!                                                5
@united thank you!                                              4
@AmericanAir thank you!                                         4
@SouthwestAir thank you!                                        3
@USAirways YOU ARE THE BEST!!! FOLLOW ME PLEASE;)🙏🙏🙏✌✌✌🙏🙏🙏    3
@USAirways thank you                                            3
@SouthwestAir Thank you!                                        3
Name: text, dtype: int64
```

Summary for Numerical Features

The primary numerical features in our data set are the tweet_id (a unique identifier - not useful to our problem), airline_sentiment_confidence and negativereason_confidence. The latter two features do have significance to our problem as they indicate the confidence in the classification of a tweet to a particular sentiment (positive, negative, neutral) and the assignment of a particular reason (root cause) to negative tweets. We may want to look at this more closley as we work on the problem and chose to remove low-confidence rows.

In [15]:
```
# describe numerical columns
df.describe().transpose()
```

Out[15]:

	count	mean	std	
tweet_id	14640.00	569218351767499200.00	779111158481835.88	567588278875
airline_sentiment_confidence	14640.00	0.90	0.16	
negativereason_confidence	10522.00	0.64	0.33	

Plots

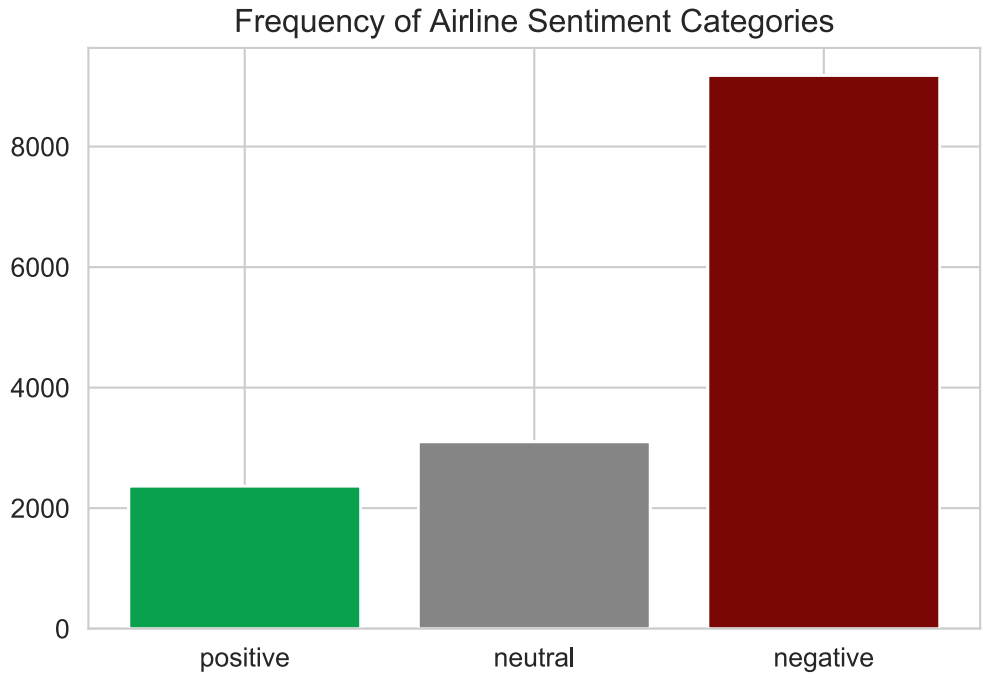
We'll examine the data set further by graphically plotting various key features.

Airline Sentiment

The 'airline_sentiment' column is a classification of whether a tweet is considered 'positive', 'neutral' or 'negative'. A positive tweet is one that says something good or desirable about the airline. The airline would want to encourage these tweets if possible. A negative tweet is a complaint or a negative sentiment about the airline. One of the stated goals of this project is to identify these so that the airline can monitor closely. A neutral tweet is text that could not be classified either way.

The plot below shows the difference in frequency of these sentiment categories. As you can see, the data set is unbalanced, having significantly more negative sentiment tweets than the other two categories. This is a problem for accurate classification - especially if the new data we with to classify is balanced. That said - given our business problem, which is primarily focused on the early identification of negative tweet trends, the unbalanced nature of the data set is less of a concern. Since we do have a large number of negative examples, we can confidently identify negative cases. We will investigate the possibility of balancing the data set further using techniques such as over/under sampling if it is warranted.

In [16]:
```
sentByAirline = pd.DataFrame(df, columns = ['airline', 'airline_sentiment'])

plt.bar(sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'positive'], \
        sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'positive'].count(), \
        alpha=0.5, color='#0aa14e')

plt.bar(sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'neutral'], \
        sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'neutral'].count(), \
        alpha=0.5, color='#858585')

plt.bar(sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'negative'], \
        sentByAirline['airline_sentiment'].loc[sentByAirline['airline_sentimen
t'] == 'negative'].count(), \
        alpha=0.5, color='#7a0606')
plt.title("Frequency of Airline Sentiment Categories")
plt.show()
```



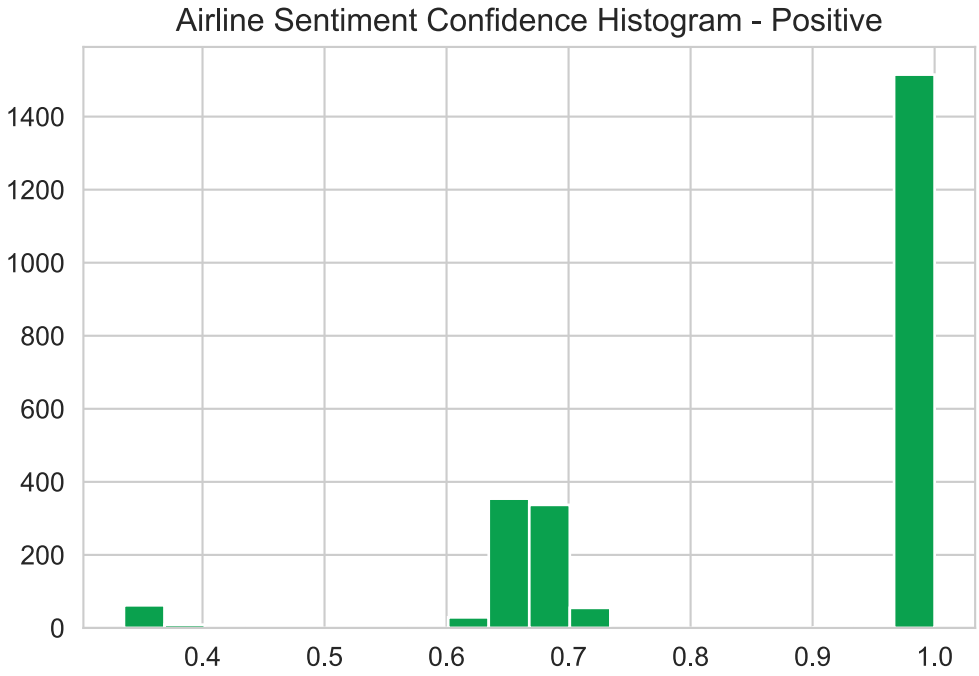
Airline Sentiment Confidence

These plots examine the distribution of another useful feature - confidence values for the choice of sentiment. A high confidence value indicates a large degree of certainty that the tweet has been classified correctly. A lower number indicates further uncertainty.

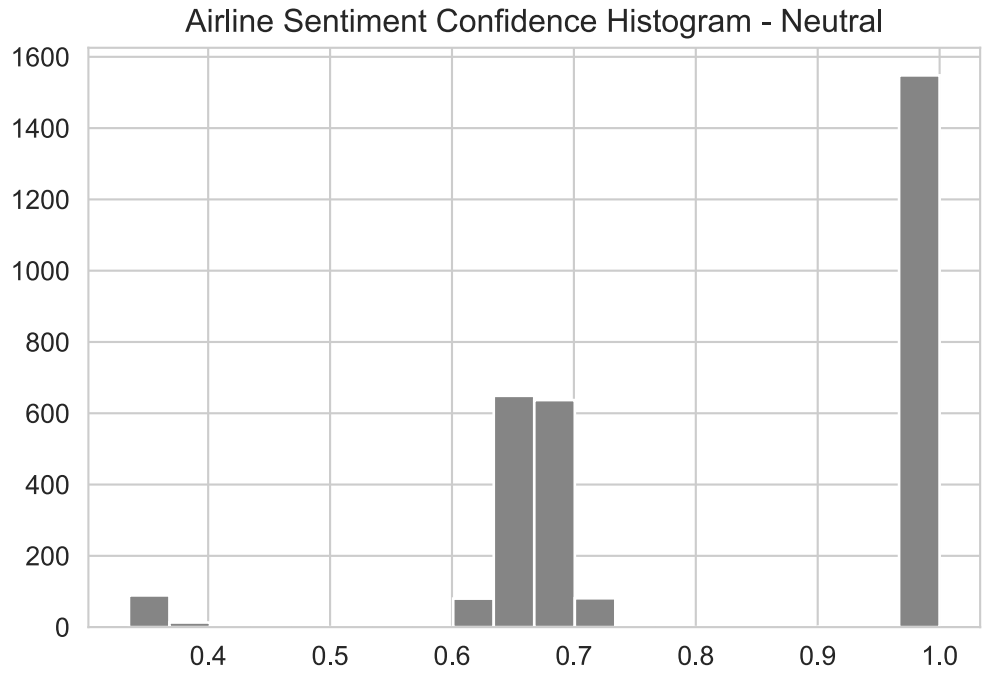
As you can see, the confidence is quite high for most of the data set, but there are some notable examples of low confidence classifications. We may want to remove the lower ranked classifications if we have sufficient data remaining for our models.

The three plots below examine the confidence value for Positive, Neutral and Negative sentiment classifications, in that order.

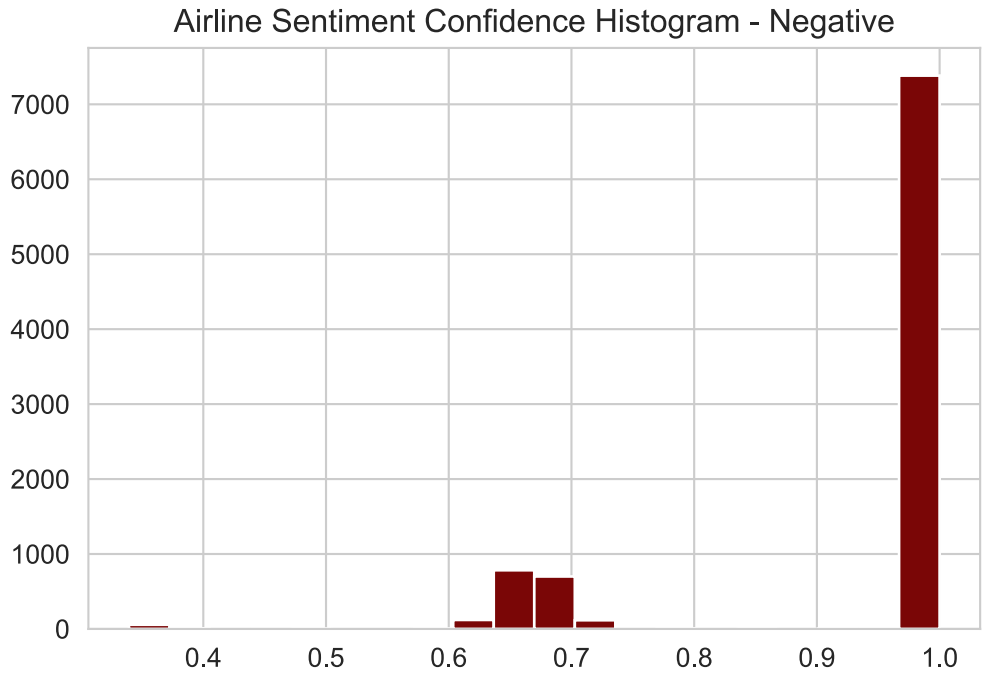
In [17]:
```
plt.hist(df['airline_sentiment_confidence'].loc[df['airline_sentiment'] == 'po
sitive'], \
         bins = 20, alpha=1, color='#0aa14e')
plt.title("Airline Sentiment Confidence Histogram - Positive")
plt.show()
```



In [18]:
```
plt.hist(df['airline_sentiment_confidence'].loc[df['airline_sentiment'] == 'neutral'], \
        bins = 20, alpha=1, color='#858585')
plt.title("Airline Sentiment Confidence Histogram - Neutral")
plt.show()
```



In [19]:
```
plt.hist(df['airline_sentiment_confidence'].loc[df['airline_sentiment'] == 'negative'], \
        bins = 20, alpha=1, color='#7a0606')
plt.title("Airline Sentiment Confidence Histogram - Negative")
plt.show()
```



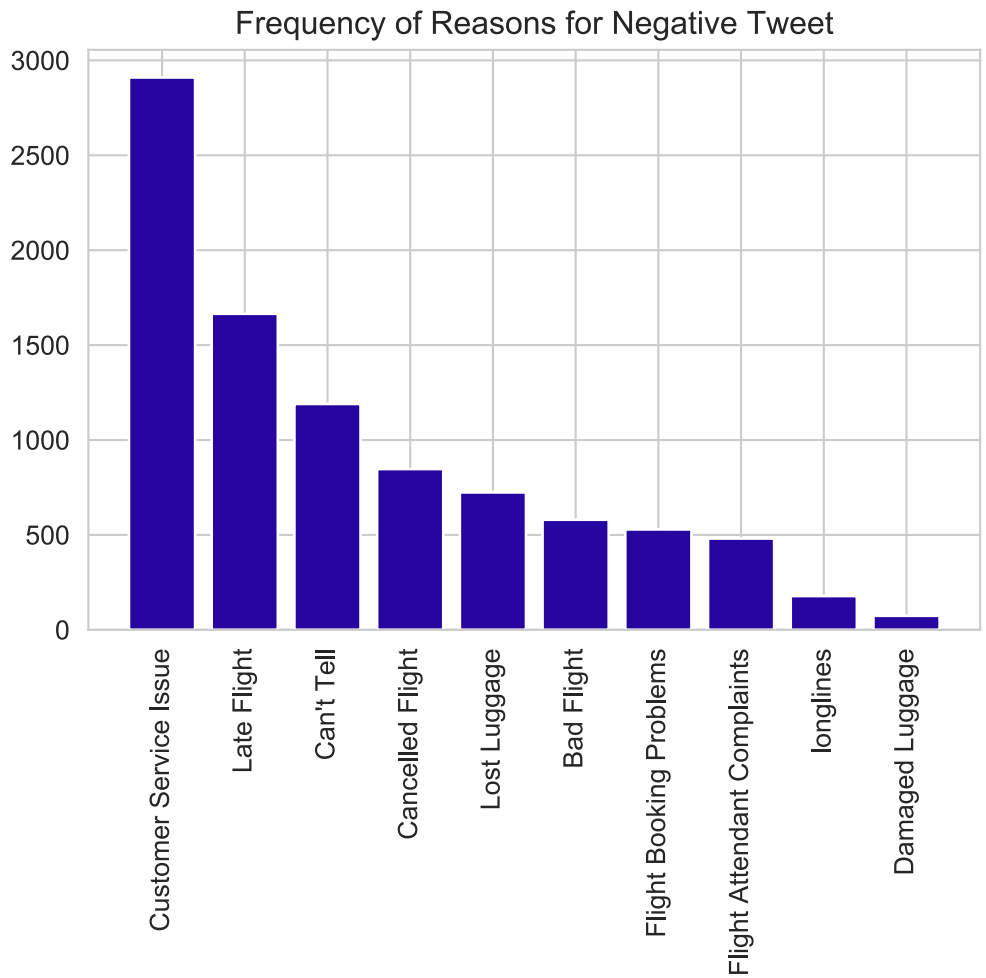
Reasons for Negative

This feature is an identified reason for the classification of a tweet as 'negative'. It required intepretation of the tweet and the assignment of a general category based on the overall problem or topic.

Having a good understanding of why people write negative tweets may help us model the data better, so it is important to explore these features.

The plot below graphs the frequency of the most common reasons for a negative sentiment tweet, in descending order.

In [20]:
```
negativeReasonCount = Counter(df['negativereason'].loc[df['airline_sentiment']
== 'negative'])
negativeReasonCount.most_common(10)

#plt.hist(df['negativereason'].loc[df['airline_sentiment'] == 'negative'], \
#         bins = 10, alpha=1, color='#2705a1')
plt.bar(*zip(*negativeReasonCount.most_common()), color='#2705a1')

plt.title("Frequency of Reasons for Negative Tweet")
plt.xticks(rotation=90)
plt.show()
```

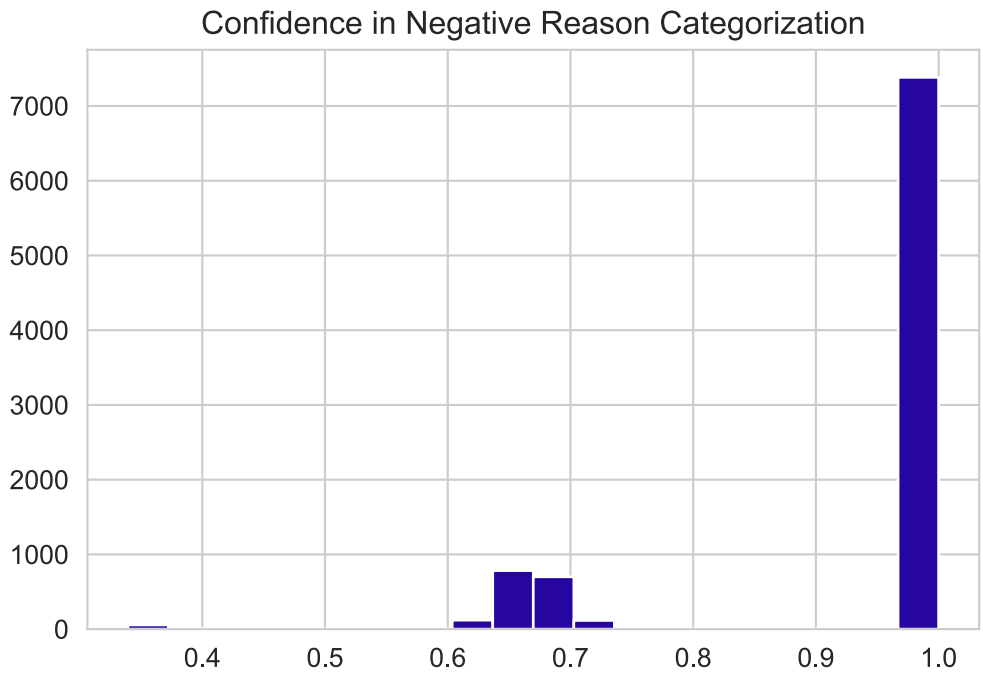


Reasons for Negative Confidence

Similar to the airline sentiment confidence value, this value indicates the degree of confidence in the assignment of a specific reason for negative sentiment being assigned to a tweet. These values are almost all above 0.6 and most sit at nearly 1.0 so based on that data this is a fairly reliable classification.

Similar to the last metric, we may want to drop any low-value outliers to ensure the accuracy of our classification.

In [21]:
```
plt.hist(df['airline_sentiment_confidence'].loc[df['airline_sentiment'] == 'ne
gative'], \
         bins = 20, alpha=1, color='#2705a1')
plt.title("Confidence in Negative Reason Categorization")
plt.show()
```

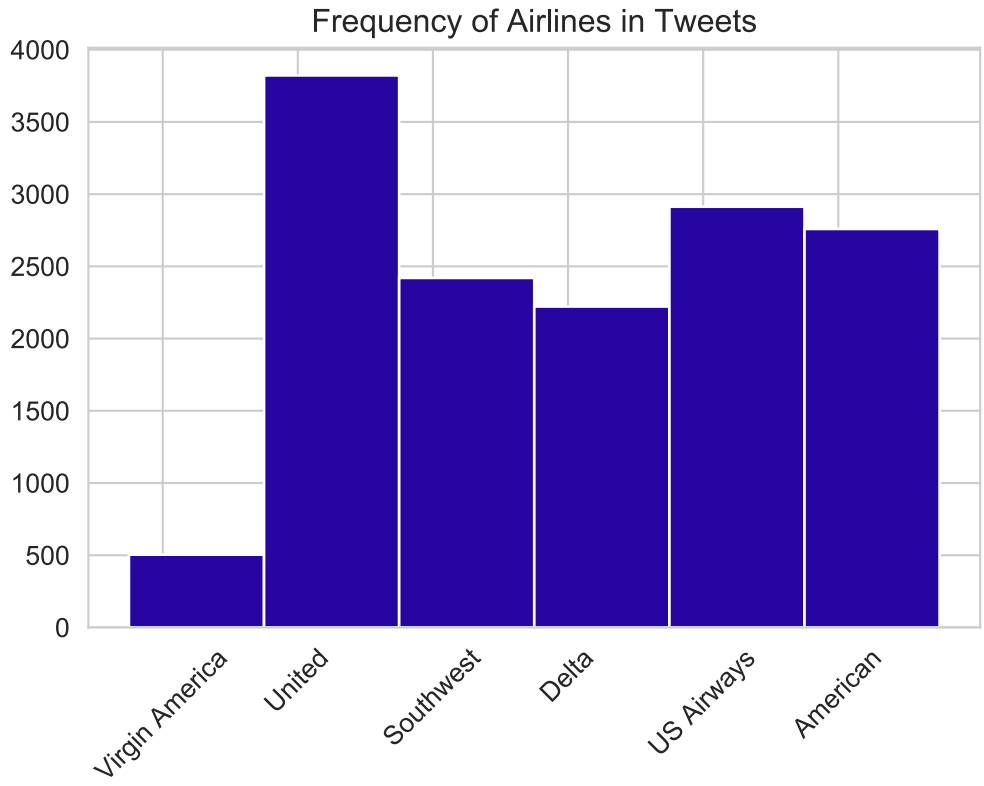


Distribution of Airline

The 'airline' feature indicates the airline that the tweet in question is referring to or directed at. We don't yet have enough data to determine how the sentiment classification will differ between airlines (if at all) but it is important to know the distribution of tweets to airlines if we choose to act on that data.

A notable difference in the plot below is 'Virgin America' which we have significantly fewer tweets for than any other airline. Further analysis may be required to determine whether this is a data set limitation, or whether it is representative of the actual market share.

In [22]:
```
plt.hist(df['airline'], \
        bins=(np.arange(7) - 0.25), alpha=1, color='#2705a1')
plt.title("Frequency of Airlines in Tweets")
plt.xticks(rotation=45)
plt.show()
```



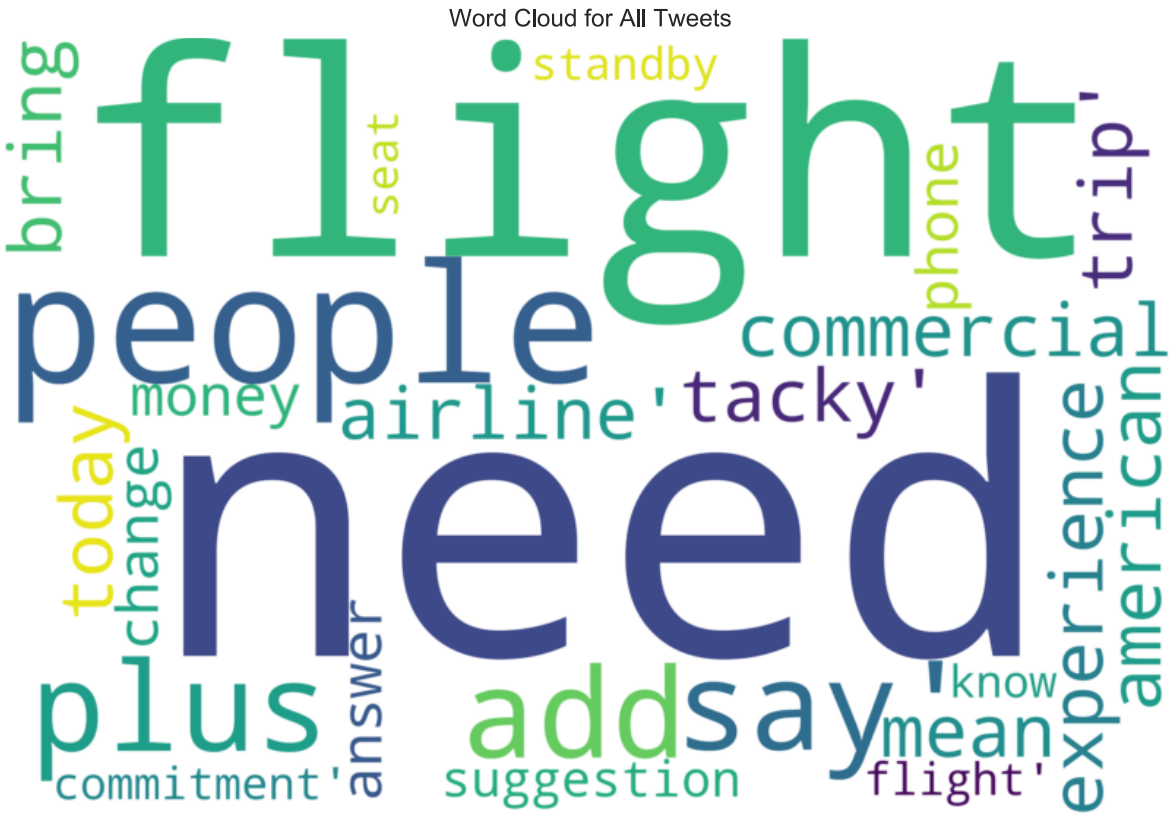
Text Analysis

In this section we'll do some basic examination of the cleaned-up text in the form of word clouds. These visualizations allow us to see both the frequency and the variations of words occuring in the available text (all of the tweets combined).

Word Cloud for All Tweets

The first word cloud has been created based on all words occurring in every tweet in the data set. In general, it should be representative of the 'average' tweet. Generic words such as 'flight', 'people', 'experience', 'trip', etc. occur as might be expected.

In [23]:
```
text = df.lemmas_list.values
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = 'white',).generate(str(text))
fig = plt.figure(
    figsize = (8, 8),
    facecolor = 'white',
    edgecolor = 'white')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Word Cloud for All Tweets")
plt.show()
```



Word Cloud for Positive Sentiment Tweets

This word cloud has been filtered down to those words used in the tweets classified as positive. As one might expect, generally positive-associate words are seen here such as 'thank', 'plus', 'pretty', etc. There are some unusual words such as 'worm' which may warrant further investigation. We will also want to investigate how strongly correlated these words are to positive sentiment. Some words are clearly recurring from the un-filtered word cloud above.

In [24]:
```
positiveText = df['lemmas_list'].loc[df['airline_sentiment'] == 'positive']
text = positiveText
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = '#DDF9DD',).generate(str(text))
fig = plt.figure(
    figsize = (8, 8),
    facecolor = 'white',
    edgecolor = 'white')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Word Cloud for Positive Sentiment Tweets")
plt.show()
```

Word Cloud for Positive Sentiment Tweets

Word Cloud for Neutral Sentiment Tweets

Though neutral tweets do make up a significant portion of our data set, they are less interesting to our business problem as they do not have a clear action to take (either damage control for a negative tweet or use for promotion/marketing on a postive tweet). It is our opinion that some neutral tweets could be classified as either positive or negative if more informatoin such as the full context of the conversation was available. However, some tweets are truly neutral such as those providing generic information.

We cannot interpret the words occuring here too closly at this time, but we hope to investigate further as we progress.

In [25]:
```
positiveText = df['lemmas_list'].loc[df['airline_sentiment'] == 'neutral']
text = positiveText
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = '#DCDCDC',).generate(str(text))
fig = plt.figure(
    figsize = (8, 8),
    facecolor = 'white',
    edgecolor = 'white')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Word Cloud for Neutral Sentiment Tweets")
plt.show()
```

Word Cloud for Negative Sentiment Tweets

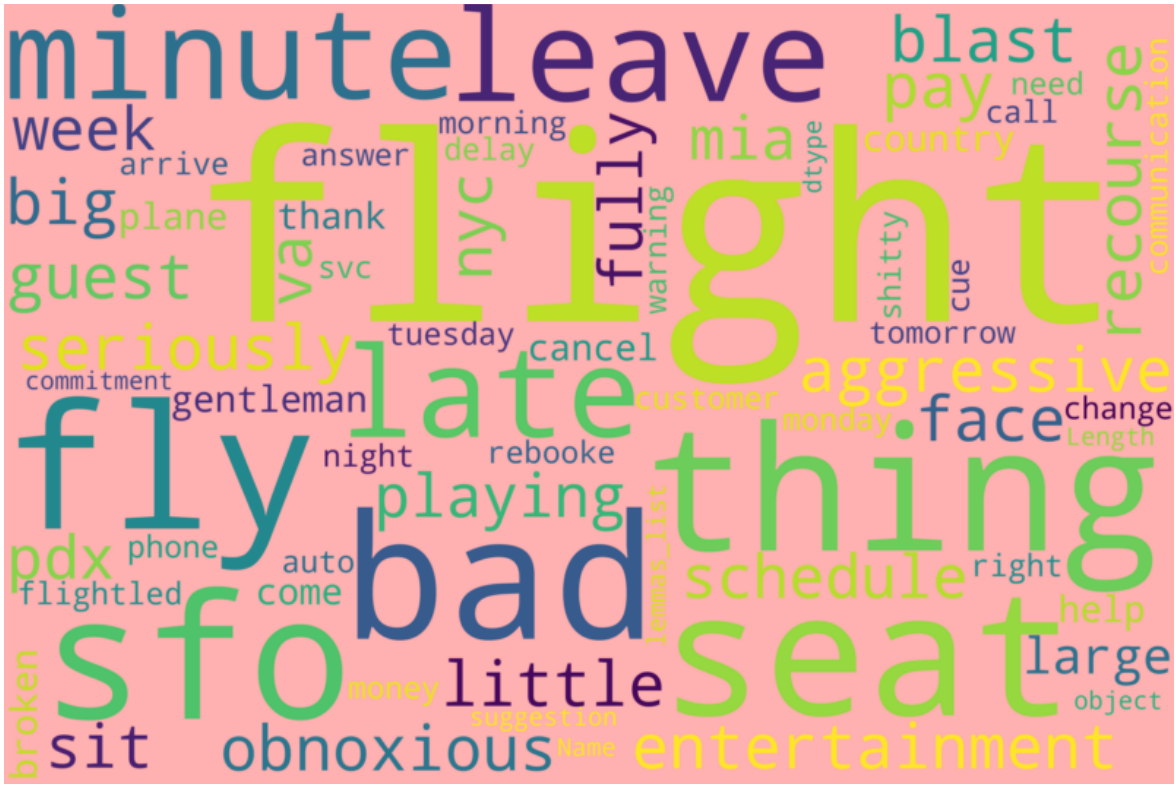
This selection of words is our most important and final batch. As our problem is primarily the identification of negative tweets, we want to pay close attention to the unique and commonly occuring words in negative tweets, as shown below.

Words such as 'bad', 'late' and 'obnoxious' all carry clear negative connotations in regards to a flight. Some words such as 'seat' and 'entertainment' require a little more interpretation to find the negative correlation. 'Seat' is likely passengers dissatisfied with their airplane seat and 'entertainment' likely refers to complains on the in-flight entertainment.

It seems that this word cloud contains more unique words than the others - this is likely becaues of the unbalanced nature of the dataset (weighting towards negative) giving us more data (more words) to work with. This will be useful to enhance our predictions on negative tweets.

In [26]:
```
positiveText = df['lemmas_list'].loc[df['airline_sentiment'] == 'negative']
text = positiveText
wordcloud = WordCloud(
    width = 3000,
    height = 2000,
    background_color = '#FFB0B0',).generate(str(text))
fig = plt.figure(
    figsize = (8, 8),
    facecolor = 'white',
    edgecolor = 'white')
plt.imshow(wordcloud, interpolation = 'bilinear')
plt.axis('off')
plt.tight_layout(pad=0)
plt.title("Word Cloud for Negative Sentiment Tweets")
plt.show()
```

Word Cloud for Negative Sentiment Tweets



Final Thoughts

It is clear to us through the various data preparation and exploration steps taken so far that there is a significant amount of useful information to be gleaned from this data set. Overall, the complexity and amount of work needed in terms of cleanup has been significantly higher than expected. The removal of various short-form English words, slang, twitter functionality, emojiis, hashtags, etc. has been significantly time consuming. It is our hope that the time devoted to this process will result in a superior model at the end of this project.

The exploration of the data has already yielded interesting results, some of which will undoubtedly confonud our classifications and some of which will significantly enhance them.

In all, we remain confident that this was a valuable data set to choose as the issues presented will serve both to enhance our skill as machine learning students and to add value to the difficult-to-produce ML model.

3. Analysis of Emojis, Emoticons, and Hashtags

This notebook covers the frequency distributions of emojis, emoticons, and hashtags across positive, neutral, and negative sentiments. The intention is to analyze and determine if there is are patterns amongst the use of emojis, emoticons, and hashtags that could possibly further help determine the sentiment of the tweet.

Import Libraries and Read the Cleaned Data

In [1]:
```
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings("ignore")
```

Read 'Tweets_cleaned.csv'

In [2]:
```
df_tweets_cleaned = pd.read_csv('..\data\Tweets_cleaned.csv', encoding='utf-8'
)
```

Separate the dataframe into positive, neutral, and negative classified sentiment.

In [3]:
```
df_tweets_positive = df_tweets_cleaned.loc[df_tweets_cleaned['airline_sentimen
t'] == 'positive']
df_tweets_neutral = df_tweets_cleaned.loc[df_tweets_cleaned['airline_sentimen
t'] == 'neutral']
df_tweets_negative = df_tweets_cleaned.loc[df_tweets_cleaned['airline_sentimen
t'] == 'negative']
```

Distribution of emojis grouped by positive, neutral, and negative sentiments

Positive tweets with Emojis

In [4]:
```
#create a list of tokens from the tweets
list_of_emojis_positive = []
df_tweets_positive_emojis = df_tweets_positive.loc[df_tweets_positive['emojis_
flag'] == True]
```

Number of tweets of positive sentiment that contained emojis:

In [5]:
```
len(df_tweets_positive_emojis)
```

Out[5]: 189

In [6]:
```
#create a list
positive_emojis = []

for i, row in df_tweets_positive_emojis.iterrows():
    tweet_emojis = df_tweets_positive_emojis.at[i, 'emojis']
    #print(type(tweet_emojis))

    tweet_emojis_list = list(tweet_emojis.split(","))

    for emoji in tweet_emojis_list:
        #print('emoji: ' + emoji)

        #strip brackets, quote, and spaces
        emoji = emoji.strip('[]')
        emoji = emoji.replace("\'", "")
        emoji = emoji.strip()

        #print('emoji_strip: ' + emoji)
        positive_emojis.append(emoji)
```

Number of emojis used in positive tweets:

In [7]:
```
len(positive_emojis)
```

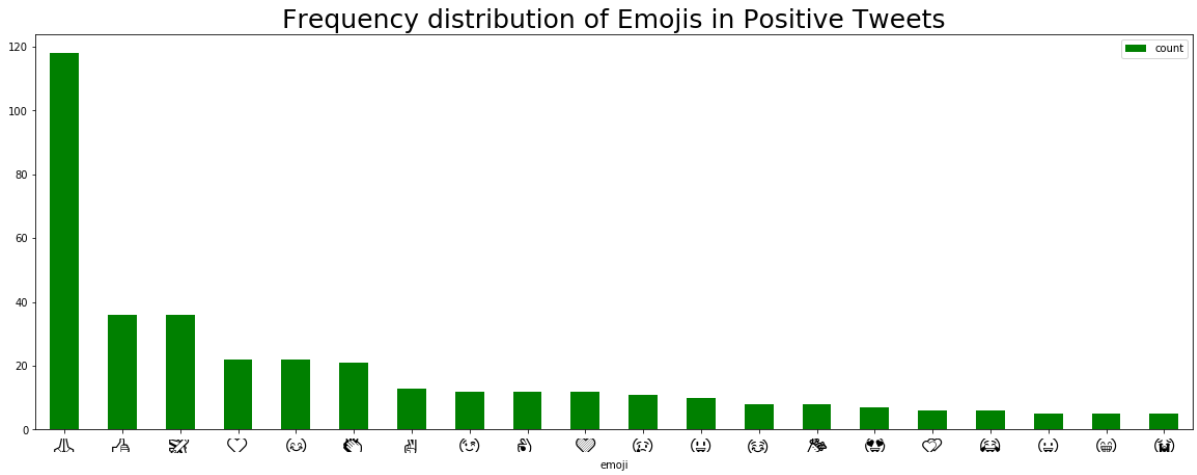
Out[7]: 457

In [8]:
```
positive_emoji_counter = Counter(positive_emojis)
positive_emoji_counter.most_common(20)
```

Out[8]:
```
[('🙏', 118),
 ('👍', 36),
 ('✈', 36),
 ('❤', 22),
 ('😊', 22),
 ('👏', 21),
 ('☝', 13),
 ('😉', 12),
 ('👌', 12),
 ('🤍', 12),
 ('😢', 11),
 ('😃', 10),
 ('☺', 8),
 ('🎉', 8),
 ('😍', 7),
 ('💞', 6),
 ('😂', 6),
 ('😀', 5),
 ('😁', 5),
 ('🤳', 5)]
```

In [9]:
```
# convert list of tuples into data frame
freq_df_positive_emoji = pd.DataFrame.from_records(positive_emoji_counter.most
_common(20),
                                   columns=['emoji', 'count'])

# create bar plot
distribution_bar_positive_emojis = freq_df_positive_emoji.plot(
    kind='bar',
    x='emoji',
    color='green',
    figsize=(20,7)
)
distribution_bar_positive_emojis.set_title(
    'Frequency distribution of Emojis in Positive Tweets',
    fontsize=25
)
distribution_bar_positive_emojis.set_xticklabels(
    freq_df_positive_emoji.emoji.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

```
Out[9]: [Text(0, 0, '🙏'),
         Text(0, 0, '👍'),
         Text(0, 0, '✈'),
         Text(0, 0, '❤'),
         Text(0, 0, '😊'),
         Text(0, 0, '👌'),
         Text(0, 0, '✌'),
         Text(0, 0, '😉'),
         Text(0, 0, '👌'),
         Text(0, 0, '♡'),
         Text(0, 0, '😘'),
         Text(0, 0, '😃'),
         Text(0, 0, '☺'),
         Text(0, 0, '🎉'),
         Text(0, 0, '😍'),
         Text(0, 0, '💕'),
         Text(0, 0, '😂'),
         Text(0, 0, '😀'),
         Text(0, 0, '😁'),
         Text(0, 0, '😭')]
```



Analysis of Frequency distribution of Emojis in Positive Tweets

Interestingly, the `Prayer Hands` emoji topped the frequency distribution. This emoji is commonly used as a way of saying "please" or "thank you", and therefore it makes sense it should be identified as an emoji with positive sentiment. All other emojis were not as commonly used. Another interesting emoji related to positive sentiment was the `Airplane` emoji; upon manual examination the text of the tweets, users sometimes tweeted this emoji as a way of expressing excitement on their trip. Surprisingly at first glance, the `Happy Face` emoji and its several variations were not used quite as often; in retrospect, this made sense simply because there is a large variety of `Happy Face` emojis available for users to use. It may be worth manually clustering the `Happy Face` emojis, as there are a large number of such categories in the lower frequency distributions.

Neutral tweets with Emojis

In [10]:
```
#create a list of tokens from the tweets
list_of_emojis_neutral = []
df_tweets_neutral_emojis = df_tweets_neutral.loc[df_tweets_neutral['emojis_fla
g'] == True]
```

Number of tweets of neutral sentiment that contained emojis:

In [11]:
```
len(df_tweets_neutral_emojis)
```

Out[11]: 127

In [12]:
```
#create a list
neutral_emojis = []

for i, row in df_tweets_neutral_emojis.iterrows():
    tweet_emojis = df_tweets_neutral_emojis.at[i, 'emojis']
    #print(type(tweet_emojis))

    tweet_emojis_list = list(tweet_emojis.split(","))

    for emoji in tweet_emojis_list:
        #print('emoji: ' + emoji)

        #strip brackets, quote, and spaces
        emoji = emoji.strip('[]')
        emoji = emoji.replace("\'", "")
        emoji = emoji.strip()

        #print('emoji_strip: ' + emoji)
        neutral_emojis.append(emoji)
```

Number of emojis used in neutral tweets:

In [13]:
```
len(neutral_emojis)
```

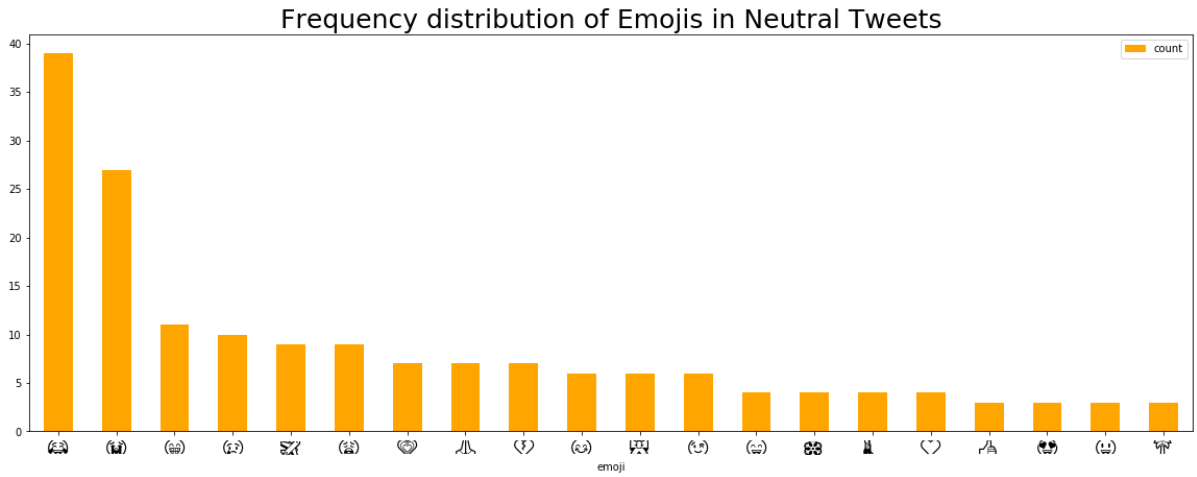
Out[13]: 239

In [14]:
```
neutral_emoji_counter = Counter(neutral_emojis)
neutral_emoji_counter.most_common(20)
```

Out[14]:
```
[('😂', 39),
 ('😭', 27),
 ('😁', 11),
 ('😢', 10),
 ('✈', 9),
 ('😩', 9),
 ('💝', 7),
 ('🙏', 7),
 ('💔', 7),
 ('😋', 6),
 ('🎀', 6),
 ('😉', 6),
 ('😄', 4),
 ('❄', 4),
 ('🎂', 4),
 ('❤', 4),
 ('👍', 3),
 ('😍', 3),
 ('😃', 3),
 ('🌴', 3)]
```

In [15]:
```
# convert list of tuples into data frame
freq_df_neutral_emoji = pd.DataFrame.from_records(neutral_emoji_counter.most_c
ommon(20),
                                    columns=['emoji', 'count'])

# create bar plot
distribution_bar_neutral_emojis = freq_df_neutral_emoji.plot(
    kind='bar',
    x='emoji',
    color='orange',
    figsize=(20,7)
)
distribution_bar_neutral_emojis.set_title(
    'Frequency distribution of Emojis in Neutral Tweets',
    fontsize=25
)
distribution_bar_neutral_emojis.set_xticklabels(
    freq_df_neutral_emoji.emoji.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

Out[15]:
```
[Text(0, 0, '😂'),
 Text(0, 0, '😭'),
 Text(0, 0, '😁'),
 Text(0, 0, '😢'),
 Text(0, 0, '✈'),
 Text(0, 0, '😩'),
 Text(0, 0, '💝'),
 Text(0, 0, '🙏'),
 Text(0, 0, '💔'),
 Text(0, 0, '😊'),
 Text(0, 0, '🎀'),
 Text(0, 0, '😉'),
 Text(0, 0, '😄'),
 Text(0, 0, '❄'),
 Text(0, 0, '🎅'),
 Text(0, 0, '❤'),
 Text(0, 0, '👍'),
 Text(0, 0, '😍'),
 Text(0, 0, '😐'),
 Text(0, 0, '👼')]
```



Analysis of Frequency distribution of Emojis in Neutral Tweets

There is much overlap of emojis used in positive and neutral tweets. For example, several of the various `Happy Face` emojis are used in both positive ane neutral tweets. On the flip side, there are only two emojis which overlap in neutral and negative sentiments. Overall,the number of emojis used in neutral tweets were far lower than positive tweets, even if there were overlaps in the emoji usage.

Negative tweets with Emojis

In [16]:
```
#create a list of tokens from the tweets
list_of_emojis_negative = []
df_tweets_negative_emojis = df_tweets_negative.loc[df_tweets_negative['emojis_
flag'] == True]
```

Number of tweets of negative sentiment that contained emojis:

In [17]:
```
len(df_tweets_negative_emojis)
```

Out[17]: 173

In [18]:
```
#create a list
negative_emojis = []

for i, row in df_tweets_negative_emojis.iterrows():
    tweet_emojis = df_tweets_negative_emojis.at[i, 'emojis']
    #print(type(tweet_emojis))

    tweet_emojis_list = list(tweet_emojis.split(","))

    for emoji in tweet_emojis_list:
        #print('emoji: ' + emoji)

        #strip brackets, quote, and spaces
        emoji = emoji.strip('[]')
        emoji = emoji.replace("\'", "")
        emoji = emoji.strip()

        #print('emoji_strip: ' + emoji)
        negative_emojis.append(emoji)
```

Number of emojis used in negative tweets:

In [19]:
```
len(negative_emojis)
```

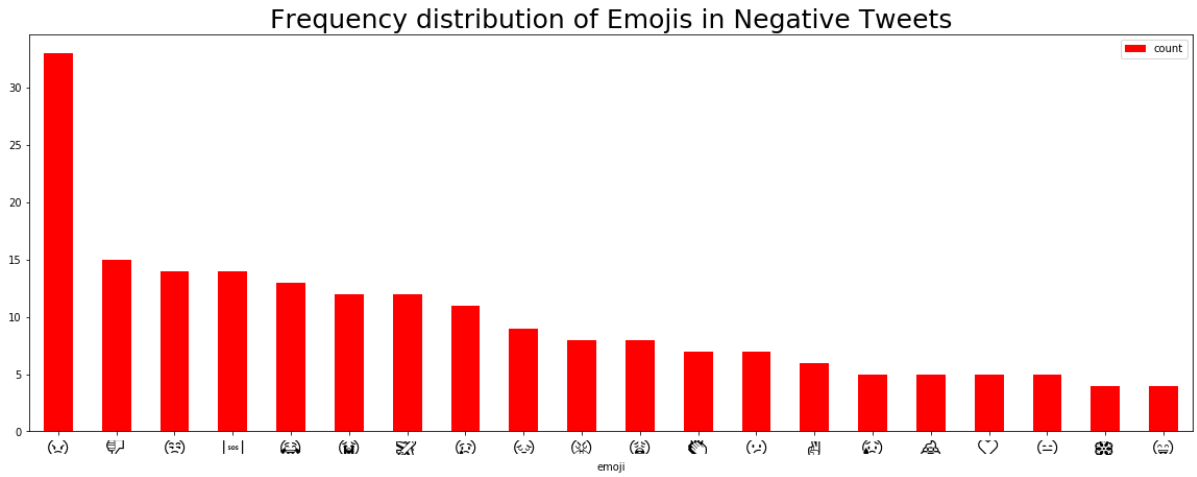
Out[19]: 280

In [20]:
```
negative_emojis_counter = Counter(negative_emojis)
negative_emojis_counter.most_common(20)
```

Out[20]:
```
[('😠', 33),
 ('👎', 15),
 ('😖', 14),
 ('🆘', 14),
 ('😂', 13),
 ('😭', 12),
 ('✈', 12),
 ('😢', 11),
 ('😔', 9),
 ('😵', 8),
 ('😩', 8),
 ('👏', 7),
 ('😕', 7),
 ('☝', 6),
 ('😥', 5),
 ('💩', 5),
 ('❤', 5),
 ('😑', 5),
 ('❄', 4),
 ('😁', 4)]
```

In [21]:
```
# convert list of tuples into data frame
freq_df_negative_emojis = pd.DataFrame.from_records(negative_emojis_counter.mo
st_common(20),
                                    columns=['emoji', 'count'])

# create bar plot
distribution_bar_negative_emojis = freq_df_negative_emojis.plot(
    kind='bar',
    x='emoji',
    color='red',
    figsize=(20,7)
)
distribution_bar_negative_emojis.set_title(
    'Frequency distribution of Emojis in Negative Tweets',
    fontsize=25
)
distribution_bar_negative_emojis.set_xticklabels(
    freq_df_negative_emojis.emoji.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

```
Out[21]: [Text(0, 0, '😠'),
          Text(0, 0, '👎'),
          Text(0, 0, '😤'),
          Text(0, 0, '🆘'),
          Text(0, 0, '😂'),
          Text(0, 0, '😭'),
          Text(0, 0, '✈'),
          Text(0, 0, '😢'),
          Text(0, 0, '😔'),
          Text(0, 0, '😵'),
          Text(0, 0, '😩'),
          Text(0, 0, '👏'),
          Text(0, 0, '😕'),
          Text(0, 0, '✌'),
          Text(0, 0, '😥'),
          Text(0, 0, '💩'),
          Text(0, 0, '❤'),
          Text(0, 0, '😑'),
          Text(0, 0, '❄'),
          Text(0, 0, '😁')]
```



Analysis of Frequency distribution of Emojis in Negative Tweets

The `Angry Face` emoji was by far the most used to express negative sentiment. The usage of the emojis indicated that users were mostly angry, sad, or dissatisfied. There is very little overlap of the emojis between negative and positive tweets, and negative and neutral tweets, but the general variety of emojis used is consistent to their usage of negative expression of sentiment. Overall number of emojis used in negative tweets was slightly higher than neutral tweets.

Distribution of emoticons grouped by positive, neutral, and negative sentiments

Positive tweets with Emoticons

In [22]:
```
#create a list of tokens from the tweets
list_of_emoticons_positive = []
df_tweets_positive_emoticons = df_tweets_positive.loc[df_tweets_positive['emot
icons_flag'] == True]
```

Number of tweets of positive sentiment that contained emoticons:

In [23]:
```
len(df_tweets_positive_emoticons)
```

Out[23]: 146

In [24]:
```
#create a list
positive_emoticons = []

for i, row in df_tweets_positive_emoticons.iterrows():
    tweet_emoticons = df_tweets_positive_emoticons.at[i, 'emoticons']
    #print(type(tweet_emoticons))

    tweet_emoticons_list = list(tweet_emoticons.split(","))

    for emoticon in tweet_emoticons_list:
        #print('emoticon: ' + emoticon)

        #strip brackets, quote, and spaces
        emoticon = emoticon.strip('[]')
        emoticon = emoticon.replace("\'", "")
        emoticon = emoticon.strip()

        #print('emoticon_strip: ' + emoticon)
        positive_emoticons.append(emoticon)
```

Number of emoticons used in positive tweets:

In [25]:
```
len(positive_emoticons)
```

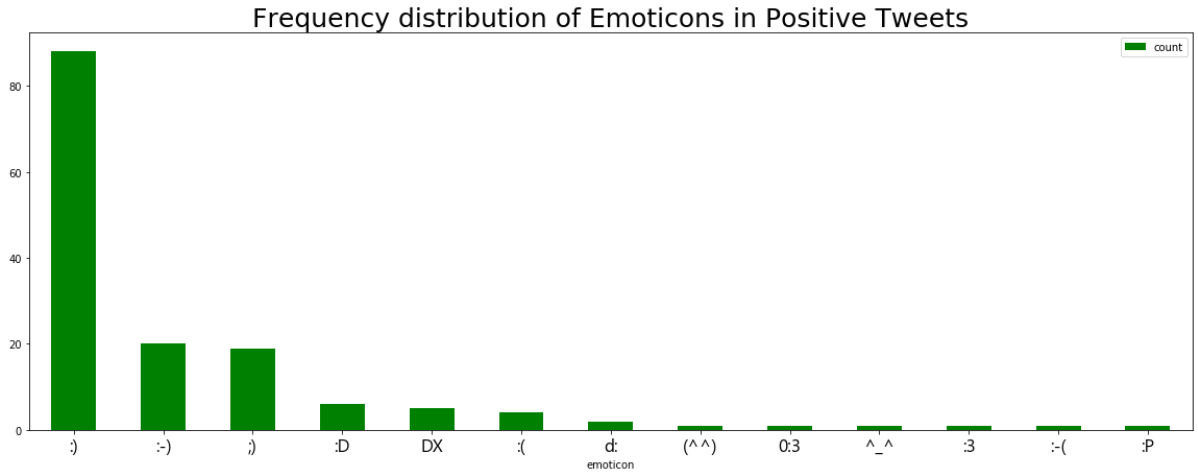
Out[25]: 150

In [26]:
```
positive_emoticons_counter = Counter(positive_emoticons)
positive_emoticons_counter.most_common(20)
```

Out[26]:
```
[(':)', 88),
 (':-)', 20),
 (';)', 19),
 (':D', 6),
 ('DX', 5),
 (':(', 4),
 ('d:', 2),
 ('(^^)', 1),
 ('0:3', 1),
 ('^_^', 1),
 (':3', 1),
 (':-(', 1),
 (':P', 1)]
```

In [27]:
```
# convert list of tuples into data frame
freq_df_positive_emoticons = pd.DataFrame.from_records(positive_emoticons_coun
ter.most_common(20),
                                       columns=['emoticon', 'count'])

# create bar plot
distribution_bar_positive_emoticons = freq_df_positive_emoticons.plot(
    kind='bar',
    x='emoticon',
    color='green',
    figsize=(20,7)
)
distribution_bar_positive_emoticons.set_title(
    'Frequency distribution of Emoticons in Positive Tweets',
    fontsize=25
)
distribution_bar_positive_emoticons.set_xticklabels(
    freq_df_positive_emoticons.emoticon.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

Out[27]:
```
[Text(0, 0, ':)'),
 Text(0, 0, ':-)'),
 Text(0, 0, ';)'),
 Text(0, 0, ':D'),
 Text(0, 0, 'DX'),
 Text(0, 0, ':('),
 Text(0, 0, 'd:'),
 Text(0, 0, '(^^)'),
 Text(0, 0, '0:3'),
 Text(0, 0, '^_^'),
 Text(0, 0, ':3'),
 Text(0, 0, ':-('),
 Text(0, 0, ':P')]
```



Analysis of Frequency distribution of Emoticons in Positive Tweets

Compared to the use of emojis in positive tweets, emoticons were not as popular. This makes sense given the "youthful" nature of Twitter, most users would know how to use emojis, instead of typing out emoticons, which is considered the "old school" method of expression emotion via pseudo-visual imagery. Unlike emojis in positive tweets, the `Happy Face` emoticon was by far the most used. Other emoticons were not used quite as frequently.

Neutral tweets with Emoticons

In [28]:
```
#create a list of tokens from the tweets
list_of_emoticons_neutral = []
df_tweets_neutral_emoticons = df_tweets_neutral.loc[df_tweets_neutral['emoticons_flag'] == True]
```

Number of tweets of neutral sentiment that contained emoticons:

In [29]:
```
len(df_tweets_neutral_emoticons)
```

Out[29]: 71

In [30]:
```
#create a list
neutral_emoticons = []

for i, row in df_tweets_neutral_emoticons.iterrows():
    tweet_emoticons = df_tweets_neutral_emoticons.at[i, 'emoticons']
    #print(type(tweet_emoticons))

    tweet_emoticons_list = list(tweet_emoticons.split(","))

    for emoticon in tweet_emoticons_list:
        #print('emoticon: ' + emoticon)

        #strip brackets, quote, and spaces
        emoticon = emoticon.strip('[]')
        emoticon = emoticon.replace("\'", "")
        emoticon = emoticon.strip()

        #print('emoticon_strip: ' + emoticon)
        neutral_emoticons.append(emoticon)
```

Number of emoticons used in neutral tweets:

In [31]:
```
len(neutral_emoticons)
```

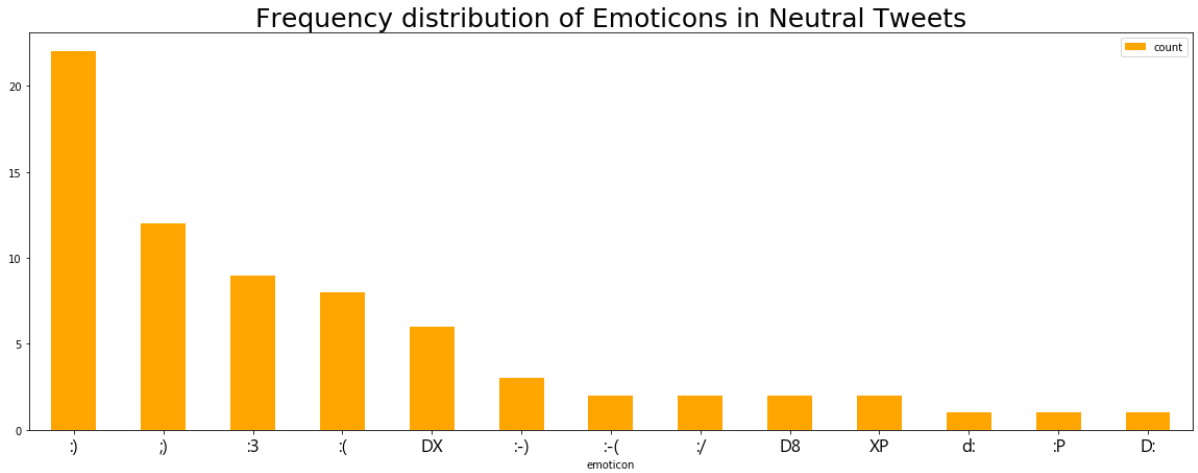
Out[31]: 71

In [32]:
```
neutral_emoticons_counter = Counter(neutral_emoticons)
neutral_emoticons_counter.most_common(20)
```

Out[32]:
```
[(':)', 22),
 (';)', 12),
 (':3', 9),
 (':(', 8),
 ('DX', 6),
 (':-)', 3),
 (':-(', 2),
 (':/', 2),
 ('D8', 2),
 ('XP', 2),
 ('d:', 1),
 (':P', 1),
 ('D:', 1)]
```

In [33]:
```
# convert list of tuples into data frame
freq_df_neutral_emoticons = pd.DataFrame.from_records(neutral_emoticons_counte
r.most_common(20),
                                     columns=['emoticon', 'count'])

# create bar plot
distribution_bar_neutral_emoticons = freq_df_neutral_emoticons.plot(
    kind='bar',
    x='emoticon',
    color='orange',
    figsize=(20,7)
)
distribution_bar_neutral_emoticons.set_title(
    'Frequency distribution of Emoticons in Neutral Tweets',
    fontsize=25
)
distribution_bar_neutral_emoticons.set_xticklabels(
    freq_df_neutral_emoticons.emoticon.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

Out[33]:
```
[Text(0, 0, ':)'),
 Text(0, 0, ';)'),
 Text(0, 0, ':3'),
 Text(0, 0, ':('),
 Text(0, 0, 'DX'),
 Text(0, 0, ':-)'),
 Text(0, 0, ':-('),
 Text(0, 0, ':/'),
 Text(0, 0, 'D8'),
 Text(0, 0, 'XP'),
 Text(0, 0, 'd:'),
 Text(0, 0, ':P'),
 Text(0, 0, 'D:')]
```



Analysis of Frequency distribution of Emoticons in Neutral Tweets

Similar to the use of emojis in neutral tweets, there is much overlap between the use of emoticons between positive and neutral sentiments, and not as much overlap between the use of emoticons between neutral and negative sentiments. There were not a lot of neutral tweets that contained emoticons either.

Negative tweets with Emoticons

In [34]:
```
#create a list of tokens from the tweets
list_of_emoticons_negative = []
df_tweets_negative_emoticons = df_tweets_negative.loc[df_tweets_negative['emot
icons_flag'] == True]
```

Number of tweets of negative sentiment that contained emoticons:

In [35]:
```
len(df_tweets_negative_emoticons)
```

Out[35]: 156

In [36]:
```
#create a list
negative_emoticons = []

for i, row in df_tweets_negative_emoticons.iterrows():
    tweet_emoticons = df_tweets_negative_emoticons.at[i, 'emoticons']
    #print(type(tweet_emoticons))

    tweet_emoticons_list = list(tweet_emoticons.split(","))

    for emoticon in tweet_emoticons_list:
        #print('emoticon: ' + emoticon)

        #strip brackets, quote, and spaces
        emoticon = emoticon.strip('[]')
        emoticon = emoticon.replace("\'", "")
        emoticon = emoticon.strip()

        #print('emoticon_strip: ' + emoticon)
        negative_emoticons.append(emoticon)
```

Number of emoticons used in negative tweets:

In [37]:
```
len(negative_emoticons)
```

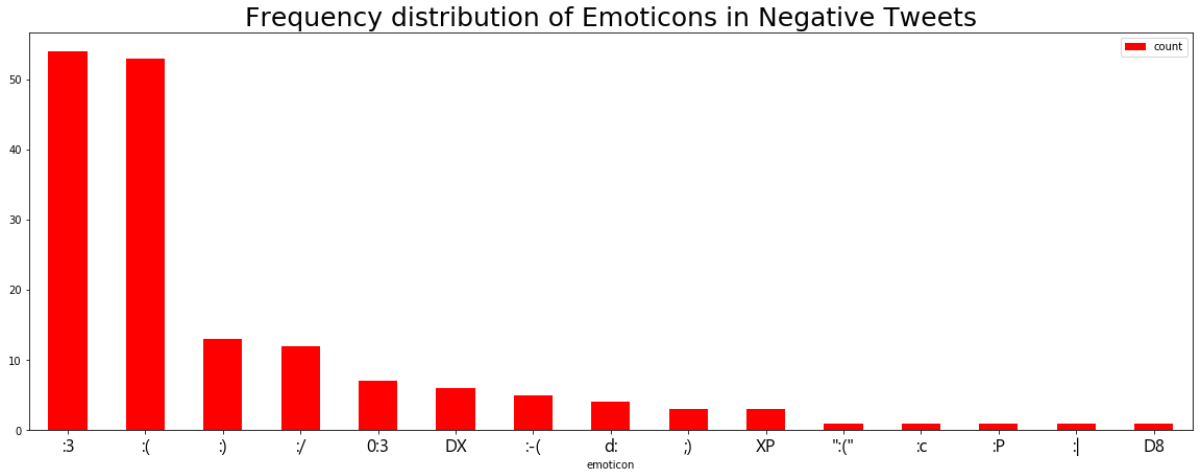
Out[37]: 165

In [38]:
```
negative_emoticons_counter = Counter(negative_emoticons)
negative_emoticons_counter.most_common(20)
```

Out[38]:
```
[(':3', 54),
 (':(', 53),
 (':)', 13),
 (':/', 12),
 ('0:3', 7),
 ('DX', 6),
 (':-(', 5),
 ('d:', 4),
 (';)', 3),
 ('XP', 3),
 ('":("', 1),
 (':c', 1),
 (':P', 1),
 (':|', 1),
 ('D8', 1)]
```

In [39]:
```
# convert list of tuples into data frame
freq_df_negative_emoticons = pd.DataFrame.from_records(negative_emoticons_coun
ter.most_common(20),
                                       columns=['emoticon', 'count'])

# create bar plot
distribution_bar_negative_emoticons = freq_df_negative_emoticons.plot(
    kind='bar',
    x='emoticon',
    color='red',
    figsize=(20,7)
)
distribution_bar_negative_emoticons.set_title(
    'Frequency distribution of Emoticons in Negative Tweets',
    fontsize=25
)
distribution_bar_negative_emoticons.set_xticklabels(
    freq_df_negative_emoticons.emoticon.tolist(),
    rotation = 0,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

Out[39]:
```
[Text(0, 0, ':3'),
 Text(0, 0, ':('),
 Text(0, 0, ':)'),
 Text(0, 0, ':/'),
 Text(0, 0, '0:3'),
 Text(0, 0, 'DX'),
 Text(0, 0, ':-('),
 Text(0, 0, 'd:'),
 Text(0, 0, ';)'),
 Text(0, 0, 'XP'),
 Text(0, 0, '":("'),
 Text(0, 0, ':c'),
 Text(0, 0, ':P'),
 Text(0, 0, ':|'),
 Text(0, 0, 'D8')]
```



Analysis of Frequency distribution of Emoticons in Negative Tweets

It must be stated that it was initially surprising to see the `Cat Face` , or `:3` emoticon top the frequency distribution. However, upon closer inspection of the data, there is a strong possibility that there was a mistake in our data cleaning steps; for some reason, tweets contained times (e.g. "12:30") would be extracted and interpreted as emoticons. This represents a data quality issue that will need to be fixed. Other emoticons with similar data quality issue include `0:3` , `DX` , and `d:` . Despite these data quality issues, it can be noted that some of the emoticons, such as the typical `Sad Face` , or `:(` emoticon do represent what is considered a normal way of expressing negative sentiment.

Distribution of hashtags grouped by positive, neutral, and negative sentiments

Positive tweets with Hashtags

In [40]:
```
#create a list of tokens from the tweets
list_of_hashtags_positive = []
df_tweets_positive_hashtags = df_tweets_positive.loc[df_tweets_positive['hasht
ags_flag'] == True]
```

Number of tweets of positive sentiment that contained hashtags:

In [41]:
```
len(df_tweets_positive_hashtags)
```

Out[41]: 435

In [42]:
```
#create a list
positive_hashtags = []

for i, row in df_tweets_positive_hashtags.iterrows():
    tweet_hashtags = df_tweets_positive_hashtags.at[i, 'hashtags']
    #print(type(tweet_hashtags))

    tweet_hashtags_list = list(tweet_hashtags.split(","))

    for hashtag in tweet_hashtags_list:
        #print('hashtag: ' + hashtag)

        #strip brackets, quote, and spaces
        hashtag = hashtag.strip('[]')
        hashtag = hashtag.replace("\'", "")
        hashtag = hashtag.strip()

        #print('hashtag_strip: ' + hashtag)
        positive_hashtags.append(hashtag)
```

Number of hashtags used in positive tweets:

In [43]:
```
len(positive_hashtags)
```

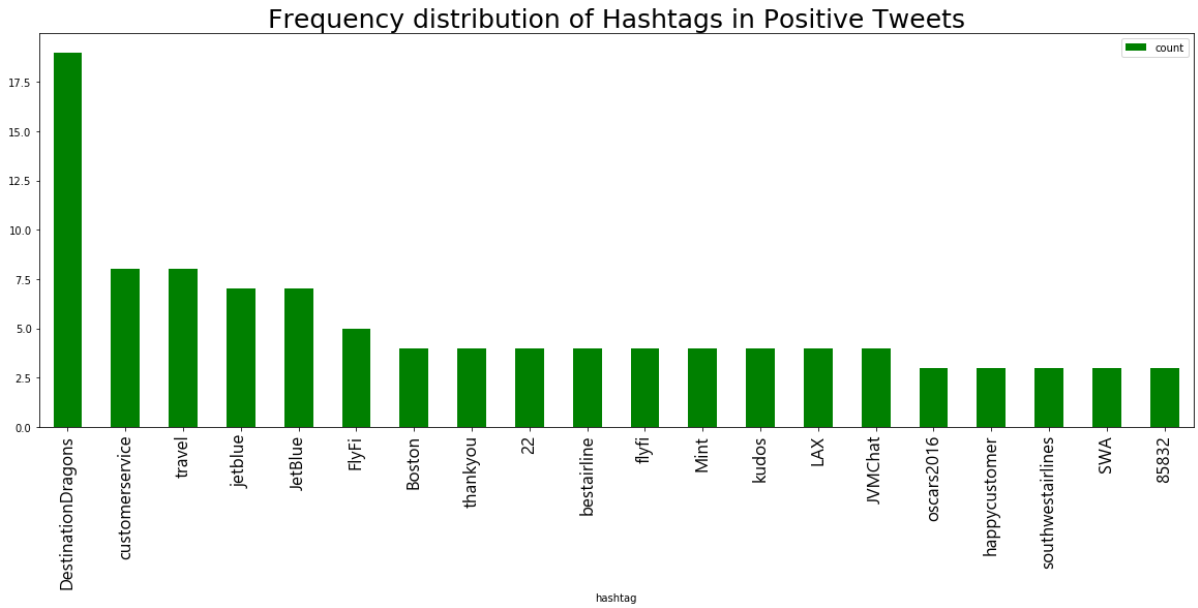
Out[43]:
```
699
```

In [44]:
```
positive_hashtags_counter = Counter(positive_hashtags)
positive_hashtags_counter.most_common(20)
```

Out[44]:
```
[('DestinationDragons', 19),
 ('customerservice', 8),
 ('travel', 8),
 ('jetblue', 7),
 ('JetBlue', 7),
 ('FlyFi', 5),
 ('Boston', 4),
 ('thankyou', 4),
 ('22', 4),
 ('bestairline', 4),
 ('flyfi', 4),
 ('Mint', 4),
 ('kudos', 4),
 ('LAX', 4),
 ('JVMChat', 4),
 ('oscars2016', 3),
 ('happycustomer', 3),
 ('southwestairlines', 3),
 ('SWA', 3),
 ('85832', 3)]
```

In [45]:
```
# convert list of tuples into data frame
freq_df_positive_hashtags = pd.DataFrame.from_records(positive_hashtags_counte
r.most_common(20),
                                      columns=['hashtag', 'count'])

# create bar plot
distribution_bar_positive_hashtags = freq_df_positive_hashtags.plot(
    kind='bar',
    x='hashtag',
    color='green',
    figsize=(20,7)
)
distribution_bar_positive_hashtags.set_title(
    'Frequency distribution of Hashtags in Positive Tweets',
    fontsize=25
)
distribution_bar_positive_hashtags.set_xticklabels(
    freq_df_positive_hashtags.hashtag.tolist(),
    rotation = 90,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

```
Out[45]: [Text(0, 0, 'DestinationDragons'),
          Text(0, 0, 'customerservice'),
          Text(0, 0, 'travel'),
          Text(0, 0, 'jetblue'),
          Text(0, 0, 'JetBlue'),
          Text(0, 0, 'FlyFi'),
          Text(0, 0, 'Boston'),
          Text(0, 0, 'thankyou'),
          Text(0, 0, '22'),
          Text(0, 0, 'bestairline'),
          Text(0, 0, 'flyfi'),
          Text(0, 0, 'Mint'),
          Text(0, 0, 'kudos'),
          Text(0, 0, 'LAX'),
          Text(0, 0, 'JVMChat'),
          Text(0, 0, 'oscars2016'),
          Text(0, 0, 'happycustomer'),
          Text(0, 0, 'southwestairlines'),
          Text(0, 0, 'SWA'),
          Text(0, 0, '85832')]
```



Analysis of Frequency distribution of Hashtags in Positive Tweets

Despite the large number of tweets with positive hashtags, the hashtags themselves were split across a large variety of hashtags, with hashtag `DestinationDragons` only appearing 19 times in 699 hashtags in 435 tweets. The hashtag `DestinationDragons` refers to the promotional event hosted by Southwest Airlines for the Imagine Dragons tour: http://destinationdragons.com/ (http://destinationdragons.com/). However, upon closer inspection, there is the possibility of another data quality issue, namely that the there were tweets with the same letters, but different case; examples include `jetblue` vs `JetBlue` , and `FlyFi` vs `flyfi` . This could possibly explain why the distribution of hashtags was seemingly distributed so equally.

Neutral tweets with Hashtags

In [46]:
```
#create a list of tokens from the tweets
list_of_hashtags_neutral = []
df_tweets_neutral_hashtags = df_tweets_neutral.loc[df_tweets_neutral['hashtags
_flag'] == True]
```

Number of tweets of neutral sentiment that contained hashtags:

In [47]:
```
len(df_tweets_neutral_hashtags)
```

Out[47]: 410

In [48]:
```
#create a list
neutral_hashtags = []

for i, row in df_tweets_neutral_hashtags.iterrows():
    tweet_hashtags = df_tweets_neutral_hashtags.at[i, 'hashtags']
    #print(type(tweet_hashtags))

    tweet_hashtags_list = list(tweet_hashtags.split(","))

    for hashtag in tweet_hashtags_list:
        #print('hashtag: ' + hashtag)

        #strip brackets, quote, and spaces
        hashtag = hashtag.strip('[]')
        hashtag = hashtag.replace("\'", "")
        hashtag = hashtag.strip()

        #print('hashtag_strip: ' + hashtag)
        neutral_hashtags.append(hashtag)
```

Number of hashtags used in neutral tweets:

In [49]:
```
len(neutral_hashtags)
```

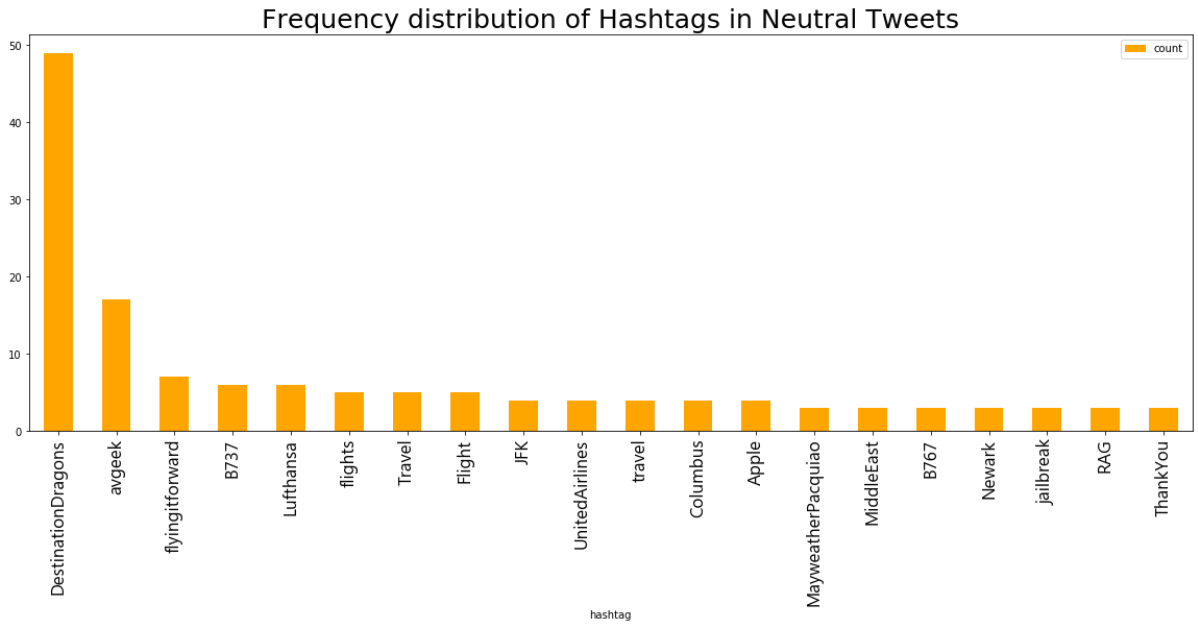
Out[49]: 653

In [50]:
```
neutral_hashtags_counter = Counter(neutral_hashtags)
neutral_hashtags_counter.most_common(20)
```

Out[50]:
```
[('DestinationDragons', 49),
 ('avgeek', 17),
 ('flyingitforward', 7),
 ('B737', 6),
 ('Lufthansa', 6),
 ('flights', 5),
 ('Travel', 5),
 ('Flight', 5),
 ('JFK', 4),
 ('UnitedAirlines', 4),
 ('travel', 4),
 ('Columbus', 4),
 ('Apple', 4),
 ('MayweatherPacquiao', 3),
 ('MiddleEast', 3),
 ('B767', 3),
 ('Newark', 3),
 ('jailbreak', 3),
 ('RAG', 3),
 ('ThankYou', 3)]
```

In [51]:
```
# convert list of tuples into data frame
freq_df_neutral_hashtags = pd.DataFrame.from_records(neutral_hashtags_counter.
most_common(20),
                                     columns=['hashtag', 'count'])

# create bar plot
distribution_bar_neutral_hashtags = freq_df_neutral_hashtags.plot(
    kind='bar',
    x='hashtag',
    color='orange',
    figsize=(20,7)
)
distribution_bar_neutral_hashtags.set_title(
    'Frequency distribution of Hashtags in Neutral Tweets',
    fontsize=25
)
distribution_bar_neutral_hashtags.set_xticklabels(
    freq_df_neutral_hashtags.hashtag.tolist(),
    rotation = 90,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

```
Out[51]: [Text(0, 0, 'DestinationDragons'),
          Text(0, 0, 'avgeek'),
          Text(0, 0, 'flyingitforward'),
          Text(0, 0, 'B737'),
          Text(0, 0, 'Lufthansa'),
          Text(0, 0, 'flights'),
          Text(0, 0, 'Travel'),
          Text(0, 0, 'Flight'),
          Text(0, 0, 'JFK'),
          Text(0, 0, 'UnitedAirlines'),
          Text(0, 0, 'travel'),
          Text(0, 0, 'Columbus'),
          Text(0, 0, 'Apple'),
          Text(0, 0, 'MayweatherPacquiao'),
          Text(0, 0, 'MiddleEast'),
          Text(0, 0, 'B767'),
          Text(0, 0, 'Newark'),
          Text(0, 0, 'jailbreak'),
          Text(0, 0, 'RAG'),
          Text(0, 0, 'ThankYou')]
```



Analysis of Frequency distribution of Hashtags in Neutral Tweets

For some reason, neutral tweets featured `DestinationDragons` more frequently at 49 occurences, compared to positive tweets featuring the same hashtag. Unfortunately, the data quality issue with hashtags mentioned earlier with positive tweets likely has affected neutral tweets as well: `travel` vs `Travel`, `flights` vs `Flight` (although this is also a case of not lemmatizing the hashtags). Something that is curious is that neutral tweets had hashtags that referenced a specific airplane model, such as `B737` and `B767`, which I find personally quite odd.

Negative tweets with Hashtags

In [52]:
```
#create a list of tokens from the tweets
list_of_hashtags_negative = []
df_tweets_negative_hashtags = df_tweets_negative.loc[df_tweets_negative['hasht
ags_flag'] == True]
```

Number of tweets of negative sentiment that contained hashtags:

In [53]:
```
len(df_tweets_negative_hashtags)
```

Out[53]: 1523

In [54]:
```
#create a list
negative_hashtags = []

for i, row in df_tweets_negative_hashtags.iterrows():
    tweet_hashtags = df_tweets_negative_hashtags.at[i, 'hashtags']
    #print(type(tweet_hashtags))

    tweet_hashtags_list = list(tweet_hashtags.split(","))

    for hashtag in tweet_hashtags_list:
        #print('hashtag: ' + hashtag)

        #strip brackets, quote, and spaces
        hashtag = hashtag.strip('[]')
        hashtag = hashtag.replace("\'", "")
        hashtag = hashtag.strip()

        #print('hashtag_strip: ' + hashtag)
        negative_hashtags.append(hashtag)
```

Number of hashtags used in negative tweets:

In [55]:
```
len(negative_hashtags)
```

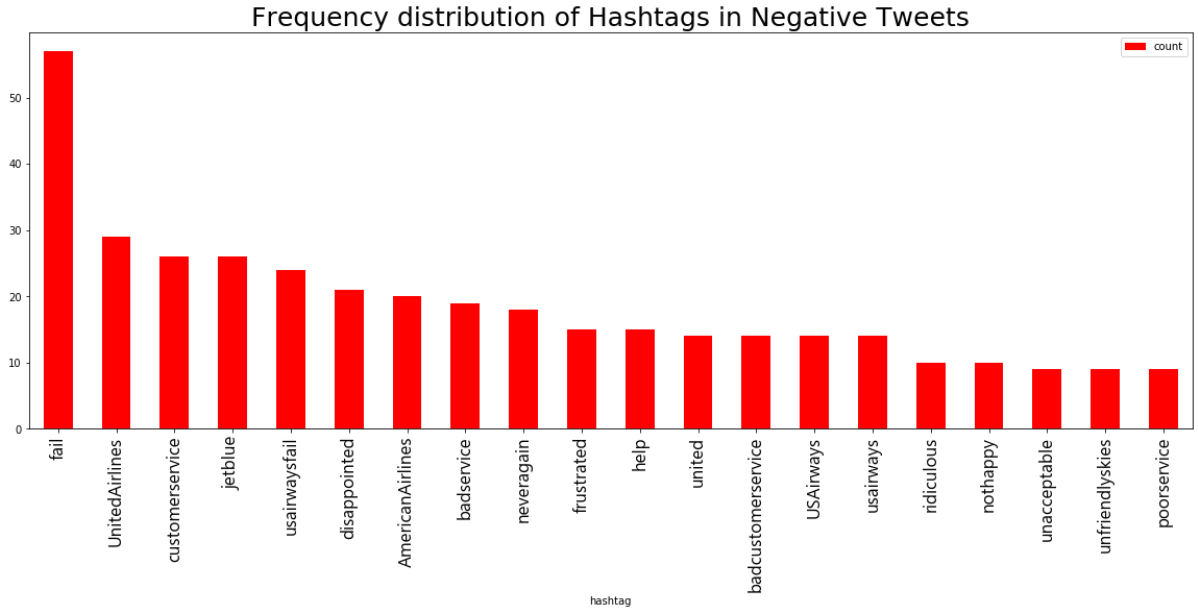
Out[55]: 2140

In [56]:
```
negative_hashtags_counter = Counter(negative_hashtags)
negative_hashtags_counter.most_common(20)
```

Out[56]:
```
[('fail', 57),
 ('UnitedAirlines', 29),
 ('customerservice', 26),
 ('jetblue', 26),
 ('usairwaysfail', 24),
 ('disappointed', 21),
 ('AmericanAirlines', 20),
 ('badservice', 19),
 ('neveragain', 18),
 ('frustrated', 15),
 ('help', 15),
 ('united', 14),
 ('badcustomerservice', 14),
 ('USAirways', 14),
 ('usairways', 14),
 ('ridiculous', 10),
 ('nothappy', 10),
 ('unacceptable', 9),
 ('unfriendlyskies', 9),
 ('poorservice', 9)]
```

In [57]:
```
# convert list of tuples into data frame
freq_df_negative_hashtags = pd.DataFrame.from_records(negative_hashtags_counte
r.most_common(20),
                                      columns=['hashtag', 'count'])

# create bar plot
distribution_bar_negative_hashtags = freq_df_negative_hashtags.plot(
    kind='bar',
    x='hashtag',
    color='red',
    figsize=(20,7)
)
distribution_bar_negative_hashtags.set_title(
    'Frequency distribution of Hashtags in Negative Tweets',
    fontsize=25
)
distribution_bar_negative_hashtags.set_xticklabels(
    freq_df_negative_hashtags.hashtag.tolist(),
    rotation = 90,
    fontsize = 17,
    #fontproperties=prop
    fontname='Segoe UI Emoji'
)
```

Out[57]:
```
[Text(0, 0, 'fail'),
 Text(0, 0, 'UnitedAirlines'),
 Text(0, 0, 'customerservice'),
 Text(0, 0, 'jetblue'),
 Text(0, 0, 'usairwaysfail'),
 Text(0, 0, 'disappointed'),
 Text(0, 0, 'AmericanAirlines'),
 Text(0, 0, 'badservice'),
 Text(0, 0, 'neveragain'),
 Text(0, 0, 'frustrated'),
 Text(0, 0, 'help'),
 Text(0, 0, 'united'),
 Text(0, 0, 'badcustomerservice'),
 Text(0, 0, 'USAirways'),
 Text(0, 0, 'usairways'),
 Text(0, 0, 'ridiculous'),
 Text(0, 0, 'nothappy'),
 Text(0, 0, 'unacceptable'),
 Text(0, 0, 'unfriendlyskies'),
 Text(0, 0, 'poorservice')]
```



Analysis of Frequency distribution of Hashtags in Negative Tweets

The number of negative tweets with hashtags was approximately double the number of positive and negative tweets combined. This itself is interesting because it seems to imply a casuality that users would use hashtags more often to express negative sentiments. There is some credence to this theory, despite the warnings of the oft-repeated phrase "Correlation does not imply causation", as users who are likely upset or sad with the airlines will want to raise this issue as quickly as possible in social media so that they are likelier to be rewarded/compensated; in Twitter, a likely way of getting a viral tweet is using as many hashtags as possible so it appears more often in multiple user's Twitter feeds. Nevertheless, this is just a theory: what was suggested is just a hypothesis, and for a hypothesis to be proven statistically significant, it must be tested.

The `fail` hashtag was used quite frequently, occuring a total of 57, but like the other most frequent hashtags in neutral and positive sentiments, it still is a small percentage of the total number of hashtags in negative tweets. Like the other positive and neutral tweets, the distribution of hashtags in negative tweets is affected by the same data quality issue: `USAirways` vs `usairways`. On top of that, another issue we can see from the frequency distribution that the same airline companies have different hashtags due to different naming conventions (for example, `united` vs `UnitedAirlines`). Solving this data quality issue will be quite challenging. Despite the aforementioned data quality issues, there are some positives to be taken away. Compared to the positive and neutral tweets, tt appears that many negative tweets contained the hashtags of the name of the airline company. Also, negative tweets also contained words normally associated with negative sentiment, such as `disappointed`, `frustrated`, `ridiculous`, `nothappy`, etc.

Final Thoughts

As discussed earlier, there were several data quality issues, particularly with the extraction of emoticons and hashtags. The issue with the extraction of emoticons is caused by a step in the cleaning process that confuses the emoticon with what is actually a non-emoticon string of characters. For example, the time and airports are misrepresented as emoticons. The data quality with the hashtags is one that consists of several interlinked problems ranging from simple to complex. These problems include making the hashtags to a single case (preferrably lower), reducing the words in the hashtag to its roots (via lemmatization, for example), and figuring out a way to group different hashtags that have the same entity, but different word.

Despite the aforementioned drawbacks, there are some positives to be gained from this analysis. There were some visible patterns in the counts and frequency distributions of emojis, emoticons, and hashtags. Some emojis and emoticons correlated well with positive or negative sentiments, with a little overlap between those sentiments with the neutral sentiment. Negative hashtags showed a clear pattern where many of the top frequent hashtags were either airline entities, or words commonly related to negative sentiment or emotion. These patterns show that emojis, emoticons, and hashtags may provide some value in classifying sentiment.

There are some additional explorations and analysis one could do further with emojis, emoticons, and hashtags:

- Clustering emojis and emoticons (e.g., see https://hal-amu.archives-ouvertes.fr/hal-01871045/document (https://hal-amu.archives-ouvertes.fr/hal-01871045/document)). A lot of emojis and emoticons share the same sentiment.
- Clustering could also work for hashtags, but is would be harder to implement because the lexicon of English words is significantly more than the lexicon of emojis/emoticons. It is not enough to just calculate string similarity between the hashtags as well using Levenshtein distance , since some entities have wildly different words.
- Exploring the median/average number of emojis/emoticons/hashtags. In particular, it appears that negative tweets seem to have more hashtags than positive or neutral tweets. We could potentially explore the number of hashtags, or even the length of a tweet to see if there is a statistically signficant correlation between them and sentiment.

In [ ]:

Feature Engineering on Twitter US Airline Dataset

Methods Used:

1. Bag of Words
2. Bag of N-Grams
3. TF-IDF
4. Cosine Document Similarity
5. word2vec trained on our dataset with tensorflow
6. word2vec trained on our dataset with gensim
7. word2vec trained by Google on News dataset

Import common packages

In [5]:
```
import pandas as pd
import numpy as np
import nltk
```

Load Cleaned Data Set

See previous code submission from project proposal for extensive data cleaning steps.

In [6]:
```
cleanDF = pd.read_csv("C:\git\CSML1010-Group_11-Final-Project\proposal\Tweets_cleaned.csv")
cleanDF.columns
```

Out[6]:
```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline', 'text',
       'text_cleaned', 'emojis_flag', 'emojis', 'emoticons_flag', 'emoticons',
       'text_cleaned_without_emojis_emoticons', 'hashtags',
       'text_cleaned_without_emojis_emoticons_hashtags', 'hashtags_flag',
       'text_cleaned_lower_case', 'text_cleaned_no_abbreviations',
       'text_list_no_stop_words', 'lemmas_list'],
      dtype='object')
```

Bag of Words Model

A bag of words model is a simpleway to represent text data as numeric vectors.

Each column is a word from our data set, each row is a single tweet and the value in each row is the number of occurrences of that word in the tweet.

In [3]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer(min_df=0., max_df=1.)
cv_matrix = cv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))
#.values.astype('U') converts the column of words to a unicode string
cv_matrix = cv_matrix.toarray()
cv_matrix
```

Out[3]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [4]:
```
# get all unique words in the corpus
vocab = cv.get_feature_names()
# show document feature vectors
pd.DataFrame(cv_matrix, columns=vocab)

#pd.options.display.max_columns = 100
#pd.set_option('display.max_rows', 100)

#pd.DataFrame(cv_matrix, columns=vocab).to_csv("bagofwords.csv")
```

Out[4]:

	00	000	000114	000lb	00a	00am	00p	00pm	01	01pm	...	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom	z
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14636	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14637	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14638	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	
14639	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	0	0	

14640 rows × 8669 columns

Bag of N-Grams

A bag of n-grams model is similar to the bag of words model, but here we extend it to include a "bi-gram" of two words. This allows us to see the number of occurrences of two-word pairs in each of our tweets as a numeric vector.

In [5]:
```
# you can set the n-gram range to 1,2 to get unigrams as well as bigrams
bv = CountVectorizer(ngram_range=(2,2))
bv_matrix = bv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))

bv_matrix = bv_matrix.toarray()
vocab = bv.get_feature_names()
pd.DataFrame(bv_matrix, columns=vocab)

#pd.DataFrame(bv_matrix, columns=vocab).to_csv("bagofngrams.csv")
```

Out[5]:

	00 27	00 bag	00 check	00 don	00 flight	00 goodwill	00 happy	00 phone	00 pm	00 say	...	zone precious	zone space	zone thank	zoom sauce	zoom scroll	zuke non	zuric b
0	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
2	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
3	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
4	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14636	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14637	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14638	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	
14639	0	0	0	0	0	0	0	0	0	0	...	0	0	0	0	0	0	

14640 rows × 62070 columns

TF-IDF Model

The TF-IDF model is a slightly more complicated way of vectorizing our text data. It combines two different ways of looking at our text. *term frequency* and *inverse document frequency*.

A more details explanation of how TF-IDF works is available elsewhere. For our purposes, it outputs scaled and normalized values which are more easily comparable and usable than bag of words-based models.

In [7]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

tv = TfidfVectorizer(min_df=0., max_df=1., use_idf=True)
tv_matrix = tv.fit_transform(cleanDF['lemmas_list'].values.astype('U'))
tv_matrix = tv_matrix.toarray()

vocab = tv.get_feature_names()
pd.DataFrame(np.round(tv_matrix, 2), columns=vocab)

#pd.DataFrame(np.round(tv_matrix, 2), columns=vocab).to_csv("tfidfmodel.csv")
```

Out[7]:

	00	000	000114	000lb	00a	00am	00p	00pm	01	01pm	...	zambia	zcc82u	zero	zig	zip	zipper	zone	zoom
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
14635	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14636	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14637	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14638	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14639	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

14640 rows × 8669 columns

Cosine Document Similarity

This measure of document similarity uses a distance function to compare the vectors of each document (aka tweet) in our TF-IDF model. A value closer to 1 indicates a more similar document and a value closer to 0 indicates a dissimilar document. The values represent the cosine angle between the vectors for each tweet.

In [8]:
```
from sklearn.metrics.pairwise import cosine_similarity

similarity_matrix = cosine_similarity(tv_matrix)
similarity_df = pd.DataFrame(similarity_matrix)
similarity_df

#similarity_df.to_csv("documentsimilarity.csv")
```

Out[8]:

	0	1	2	3	4	5	6	7	8	9	...	14630	14631	14632	14633	14634	14635	
0	1.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.0(
1	0.0	1.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.0(
2	0.0	0.0	1.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.084757	0.0	0.000000	0.0(
3	0.0	0.0	0.000000	1.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.0(
4	0.0	0.0	0.000000	0.0	1.0	0.307344	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.0(
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14635	0.0	0.0	0.000000	0.0	0.0	0.028866	0.0	0.0	0.0	0.0	...	0.30427	0.188921	0.0	0.052285	0.0	1.000000	0.0
14636	0.0	0.0	0.000000	0.0	0.0	0.026716	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.098288	0.0	0.046479	1.0(
14637	0.0	0.0	0.000000	0.0	0.0	0.000000	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.000000	0.0	0.000000	0.0(
14638	0.0	0.0	0.000000	0.0	0.0	0.019337	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.035024	0.0	0.033641	0.0
14639	0.0	0.0	0.114465	0.0	0.0	0.116877	0.0	0.0	0.0	0.0	...	0.00000	0.000000	0.0	0.130844	0.0	0.070611	0.0(

14640 rows × 14640 columns

Word2Vec Word Embedding Model

Training with CBOW (Continuous Bag of Words)

Now, we look at a more advanced model. Here, we implement a word embedding model. Word embedding is a way of representing individual words in such a way that words that are similar are represented similarly.

We do this using a word2vec model which represents words as multi-dimensional vectors (in this case 100 dimensions).

We build a model from scratch here using *keras* and *tensorflow* to build a *Continuous Bag of Words* (CBOW) model. A CBOW takes multiple words that surround a target word as input, and predicts the target word based on that input.

Build Vocabulary

In [9]:
```
from keras.preprocessing import text
from keras.utils import np_utils
from keras.preprocessing import sequence

tokenizer = text.Tokenizer()
tokenizer.fit_on_texts(cleanDF['lemmas_list'].values.astype('U'))
word2id = tokenizer.word_index

word2id['PAD'] = 0
id2word = {v:k for k, v in word2id.items()}
wids = [[word2id[w] for w in text.text_to_word_sequence(doc)] for doc in cleanDF['lemmas_list'].values
.astype('U')]

vocab_size = len(word2id)
embed_size = 100
window_size = 2

print('Vocabulary Size:', vocab_size)
print('Vocabulary Sample:', list(word2id.items())[:10])
```

```
Using TensorFlow backend.
```

```
Vocabulary Size: 8872
Vocabulary Sample: [('flight', 1), ('thank', 2), ('hour', 3), ('cancel', 4), ('service', 5), ('time',
6), ('delay', 7), ('customer', 8), ('help', 9), ('get', 10)]
```

Build (context_words, target_word) pair generator

In [10]:
```
def generate_context_word_pairs(corpus, window_size, vocab_size):
    context_length = window_size*2
    for words in corpus:
        sentence_length = len(words)
        for index, word in enumerate(words):
            context_words = []
            label_word   = []
            start = index - window_size
            end = index + window_size + 1

            context_words.append([words[i]
                                  for i in range(start, end)
                                  if 0 <= i < sentence_length
                                  and i != index])
            label_word.append(word)

            x = sequence.pad_sequences(context_words, maxlen=context_length)
            y = np_utils.to_categorical(label_word, vocab_size)
            yield (x, y)
```

In [11]:
```
i = 0
for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab_size):
    if 0 not in x[0]:
        print('Context (X):', [id2word[w] for w in x[0]], '-> Target (Y):', id2word[np.argwhere(y[0])[
0][0]])

        if i == 10:
            break
        i += 1
```

```
Context (X): ['plus', 'add', 'experience', 'tacky'] -> Target (Y): commercial
Context (X): ['aggressive', 'blast', 'entertainment', 'guest'] -> Target (Y): obnoxious
Context (X): ['blast', 'obnoxious', 'guest', 'face'] -> Target (Y): entertainment
Context (X): ['obnoxious', 'entertainment', 'face', 'little'] -> Target (Y): guest
Context (X): ['entertainment', 'guest', 'little', 'recourse'] -> Target (Y): face
Context (X): ['seriously', 'pay', 'seat', 'playing'] -> Target (Y): flight
Context (X): ['pay', 'flight', 'playing', 'bad'] -> Target (Y): seat
Context (X): ['flight', 'seat', 'bad', 'thing'] -> Target (Y): playing
Context (X): ['seat', 'playing', 'thing', 'fly'] -> Target (Y): bad
Context (X): ['playing', 'bad', 'fly', 'va'] -> Target (Y): thing
Context (X): ['yes', 'nearly', 'fly', 'vx'] -> Target (Y): time
```

Build CBOW Deep Network Model

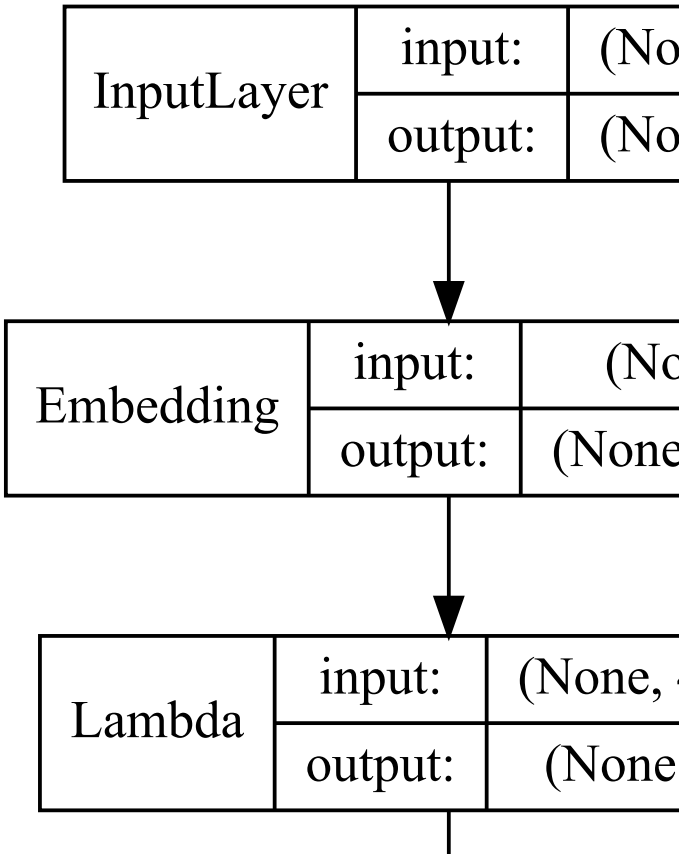
In [12]:
```
import keras.backend as K
from keras.models import Sequential
from keras.layers import Dense, Embedding, Lambda

import tensorflow as tf
with tf.device('/gpu:0'):

    cbow = Sequential()
    cbow.add(Embedding(input_dim=vocab_size, output_dim=embed_size, input_length=window_size*2))
    cbow.add(Lambda(lambda x: K.mean(x, axis=1), output_shape=(embed_size,)))
    cbow.add(Dense(vocab_size, activation='softmax'))

    cbow.compile(loss='categorical_crossentropy', optimizer='rmsprop')
    print(cbow.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 4, 100)            887200
_________________________________________________________________
lambda_1 (Lambda)            (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 8872)              896072
=================================================================
Total params: 1,783,272
Trainable params: 1,783,272
Non-trainable params: 0
_________________________________________________________________
None
```

Visualize Model

In [13]:
```
from IPython.display import SVG
from keras.utils.vis_utils import model_to_dot

SVG(model_to_dot(cbow, show_shapes=True, show_layer_names=False,
                 rankdir='TB').create(prog='dot', format='svg'))
```

Out[13]:



Train model for 5 epochs

In [ ]:
```
with tf.device('/gpu:0'):
    for epoch in range(1, 6):
        loss = 0.
        i = 0
        for x, y in generate_context_word_pairs(corpus=wids, window_size=window_size, vocab_size=vocab
_size):
            i += 1
            loss += cbow.train_on_batch(x, y)
            if i % 100000 == 0:
                print('Processed {} (context, word) pairs'.format(i))

        print('Epoch:', epoch, '\tLoss:', loss)
        print()
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\tensorflow_core\python\framework\indexed_s
lices.py:433: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. This m
ay consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

Processed 100000 (context, word) pairs
Epoch: 1        Loss: 980699.2105463557

Processed 100000 (context, word) pairs
Epoch: 2        Loss: 1144282.866285345

Processed 100000 (context, word) pairs
Epoch: 3        Loss: 1337395.7755461363

Processed 100000 (context, word) pairs
Epoch: 4        Loss: 1453879.772978104
```

Get word embeddings

In [30]:
```
#load weights (COMMENT OUT BELOW if re-training)
weights = pd.read_csv("word2vecCBOWtrained.csv")
weights = cbow.get_weights()[0]
weights = weights[1:]

print(weights.shape)



#word2vecCBOWtrained = pd.DataFrame(weights, index=list(id2word.values())[1:]).head()

#pd.DataFrame(weights, index=list(id2word.values())[1:]).to_csv("word2vecCBOWtrained.csv")
```

```
(8871, 100)
```

Build a distance matrix to view the most similar words (contextually)¶

In [40]:
```
#load word embeddings (COMMENT OUT BELOW if re-training)
word2vecCBOWtrained = pd.read_csv("word2vecCBOWtrained.csv")

from sklearn.metrics.pairwise import euclidean_distances

# compute pairwise distance matrix
distance_matrix = euclidean_distances(weights)
print(distance_matrix.shape)

# view contextually similar words
similar_words = {search_term: [id2word[idx] for idx in distance_matrix[word2id[search_term]-1].argsort
()[1:6]+1]
                 for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}

similar_words
```

```
(8871, 8871)
```

Out[40]:
```
{'flight': ['7hours', 'functionality', 'tvs', 'us2065', 'typical'],
 'airline': ['rocky', 'reduction', 'squeak', 'dedicated', 'prohibit'],
 'good': ['dnx58v', 'fog', 'fax', 'rr', '♡'],
 'bad': ['ua1266', 'win', 'lekvhg', 'crucial', 'underserved'],
 'time': ['practice', 'm', 'aa3230', 'wifi', 'loading'],
 'seat': ['taiwan', 'bogota', 'brendan', 'fudgin', 'yaayy'],
 'amazing': ['steve', "nat'l", '24hours', 'row7', 'eyewitness'],
 'experience': ['jean', 'any1', 'attitude', 'baby', 'hayes']}
```

Visualize word embeddings

In [44]:
```
word2vecCBOWtrained.iloc[:, 0]
```

Out[44]:
```
0            thank
1             hour
2           cancel
3          service
4             time
          ...
8823        stuffy
8824    arbitrarily
8825    retribution
8826          aires
8827            PAD
Name: Unnamed: 0, Length: 8828, dtype: object
```

Word2Vec Word Embedding Model

Using gensim to train model

Now, to give us another feature to compare with, we will generate a similar word2vec model, using the Continuous Bag of Words model - this time, using gensim. We are again training this on our own data set, so we expect similar results. It will be a good double-check that both models are performing similarly.

In [46]:
```
from gensim.models import word2vec

# tokenize sentences in corpus
wpt = nltk.WordPunctTokenizer()
tokenized_corpus = [wpt.tokenize(document) for document in (cleanDF['lemmas_list'].values.astype('U'
))]

# Set values for various parameters
feature_size = 100    # Word vector dimensionality
window_context = 30          # Context window size
min_word_count = 1   # Minimum word count
sample = 1e-3   # Downsample setting for frequent words

w2v_model = word2vec.Word2Vec(tokenized_corpus, size=feature_size,
                          window=window_context, min_count=min_word_count,
                          sample=sample, iter=50)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                 for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}
similar_words
```

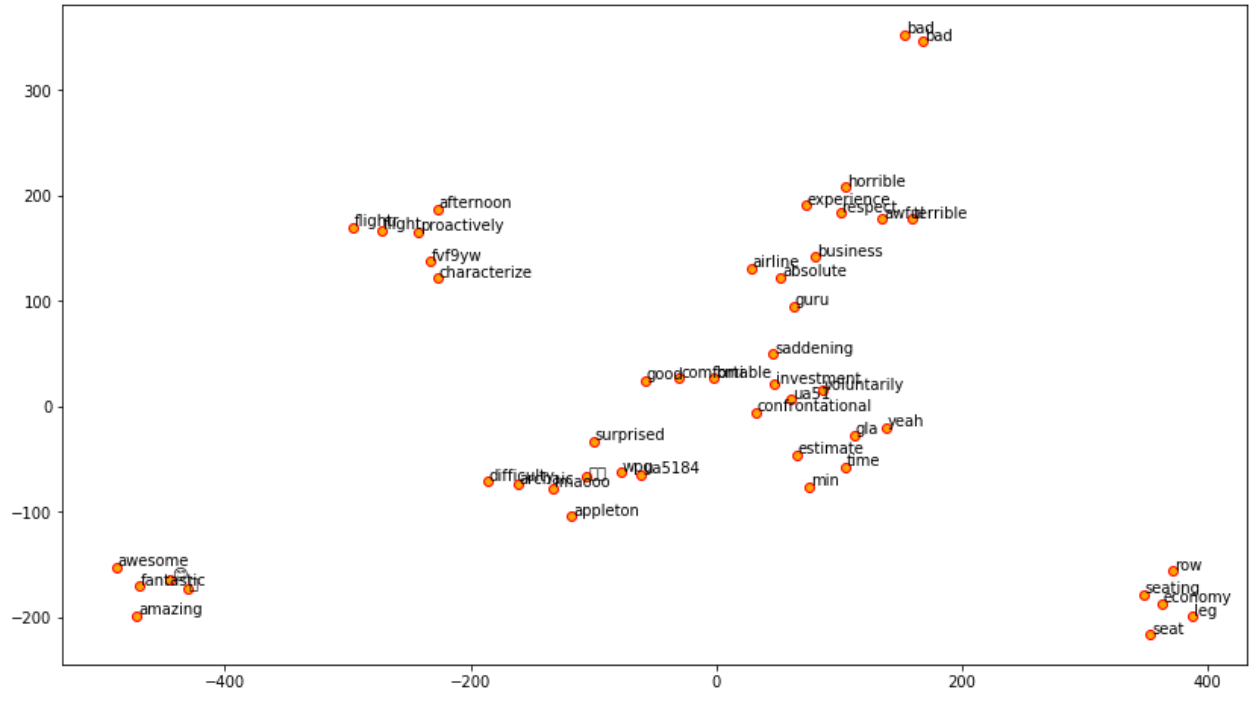
Out[46]:
```
{'flight': ['afternoon', 'flightr', 'proactively', 'fvf9yw', 'characterize'],
 'airline': ['saddening', 'investment', 'business', 'absolute', 'bmi'],
 'good': ['comfortable', 'appleton', 'surprised', 'lmaooo', 'bad'],
 'bad': ['horrible', 'terrible', 'awful', 'difficulty', 'archaic'],
 'time': ['min', 'ua51', 'gla', 'yeah', 'estimate'],
 'seat': ['economy', 'seating', 'leg', 'row', 'voluntarily'],
 'amazing': ['fantastic', '👍', '😊', 'awesome', 'ہوہو'],
 'experience': ['ua5184', 'confrontational', 'wpg', 'respect', 'guru']}
```

In [47]:
```
from gensim.models import KeyedVectors

#export model
#w2v_model.wv.save_word2vec_format(fname='gensim_word2vec_trained.bin', binary=True)
```

Visualize word embeddings

In [48]:
```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 55357 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 56397 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:176: Ru
ntimeWarning: Glyph 128077 missing from current font.
  font.load_char(ord(s), flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 56842 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:211: Ru
ntimeWarning: Glyph 10548 missing from current font.
  font.set_text(s, 0.0, flags=flags)
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\matplotlib\backends\backend_agg.py:180: Ru
ntimeWarning: Glyph 10548 missing from current font.
  font.set_text(s, 0, flags=flags)
```



Word2Vec Word Embedding Model

Using Pre-Trained Model from Google

Subet of Google's word2vec model for the 500,000 most frequent words

For our final model, instead of training our own word2vec model, we instead download Google's pre-trained word2vec model. Google has trained this model on articles from its news service and it includes 300 dimensions in almost 3.7 GB. This is not a manageable size, but we still want to load Google's model to compare to our custom-trained models.

To make this manageable, when loading the model we limit it to the first 500,000 rows. Google's model is organized with the most frequent words at the beginning of the dataset so this should be an effective reduction.

We save this reduced model to use in our benchmarking.

In [49]:
```
from gensim.models import KeyedVectors

w2v_model = KeyedVectors.load_word2vec_format(r'C:\Users\Patrick\gensim-data\word2vec-google-news-300
\word2vec-google-news-300\GoogleNews-vectors-negative300.bin', binary=True, limit=500000)

# view similar words based on gensim's model
similar_words = {search_term: [item[0] for item in w2v_model.wv.most_similar([search_term], topn=5)]
                 for search_term in ['flight', 'airline', 'good', 'bad', 'time', 'seat', 'amazing',
'experience']}
similar_words
```

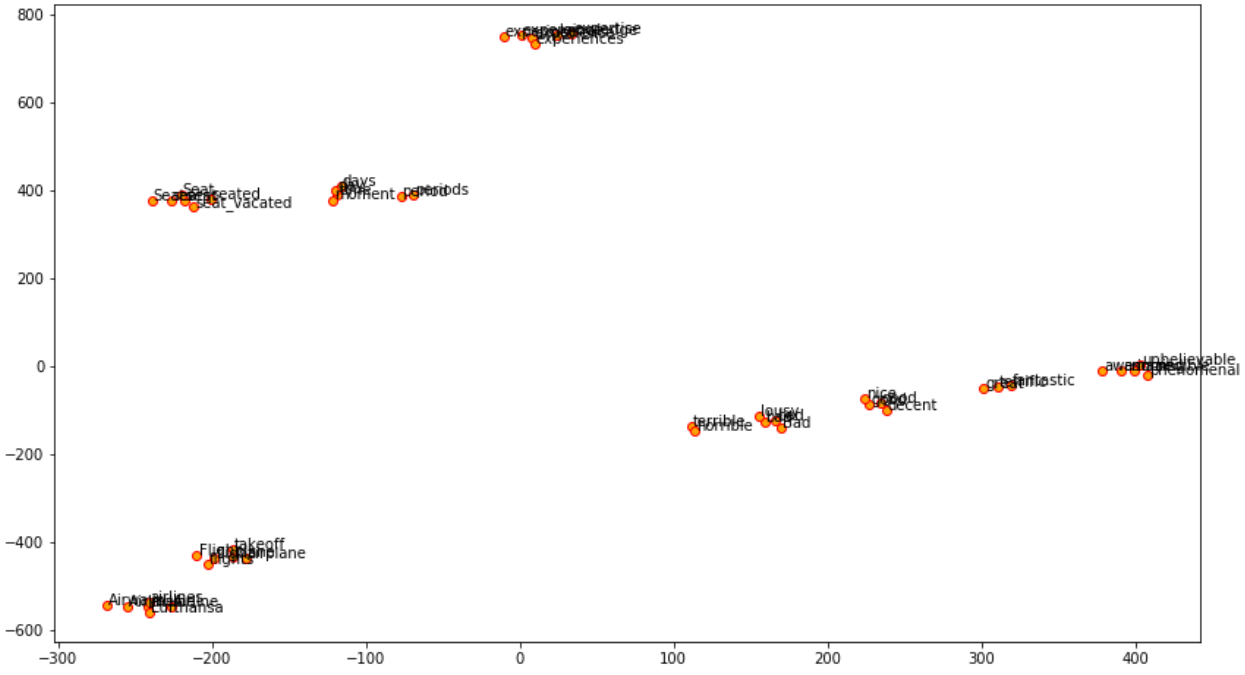
```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\ipykernel_launcher.py:7: DeprecationWarnin
g: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  import sys
```

Out[49]:
```
{'flight': ['flights', 'plane', 'Flight', 'airplane', 'takeoff'],
 'airline': ['airlines', 'Airlines', 'Airline', 'Airways', 'Lufthansa'],
 'good': ['great', 'bad', 'terrific', 'decent', 'nice'],
 'bad': ['good', 'terrible', 'horrible', 'Bad', 'lousy'],
 'time': ['day', 'moment', 'days', 'period', 'periods'],
 'seat': ['seats', 'Seat', 'Seats', 'seat_vacated', 'seated'],
 'amazing': ['incredible',
  'awesome',
  'unbelievable',
  'fantastic',
  'phenomenal'],
 'experience': ['experiences',
  'expertise',
  'expereince',
  'experince',
  'knowledge']}
```

In [50]:
```
from gensim.models import KeyedVectors

#export model
#w2v_model.wv.save_word2vec_format(fname='gensim_word2vec_google_subset.bin', binary=True)
```

In [51]:
```
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

words = sum([[k] + v for k, v in similar_words.items()], [])
wvs = w2v_model.wv[words]

tsne = TSNE(n_components=2, random_state=0, n_iter=10000, perplexity=2)
np.set_printoptions(suppress=True)
T = tsne.fit_transform(wvs)
labels = words

plt.figure(figsize=(14, 8))
plt.scatter(T[:, 0], T[:, 1], c='orange', edgecolors='r')
for label, x, y in zip(labels, T[:, 0], T[:, 1]):
    plt.annotate(label, xy=(x+1, y+1), xytext=(0, 0), textcoords='offset points')
```

```
C:\ProgramData\Anaconda3\envs\milestone1\lib\site-packages\ipykernel_launcher.py:5: DeprecationWarnin
g: Call to deprecated `wv` (Attribute will be removed in 4.0.0, use self instead).
  """
```



In [ ]:
```
```

General Testing

In [52]:
```
#check if using GPU
import tensorflow as tf
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")
```

```
Default GPU Device: /device:GPU:0
```

In [ ]:
```
#test execution on GPU
import tensorflow as tf
with tf.device('/gpu:0'):
    a = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[2, 3], name='a')
    b = tf.constant([1.0, 2.0, 3.0, 4.0, 5.0, 6.0], shape=[3, 2], name='b')
    c = tf.matmul(a, b)

with tf.Session() as sess:
    print (sess.run(c))
```

Reference:

Much of the example code is credited to Dipanjan (DJ) Sarkar at:

https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41 (https://towardsdatascience.com/understanding-feature-engineering-part-3-traditional-methods-for-text-data-f6f7d70acd41)

and

https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa (https://towardsdatascience.com/understanding-feature-engineering-part-4-deep-learning-methods-for-text-data-96c44370bbfa)

5. Feature Engineering, Feature Selection, and Baseline Benchmark

In [1]:
```
import pandas as pd
from collections import Counter
import matplotlib.pyplot as plt
import warnings
import numpy as np

warnings.filterwarnings("ignore")
pd.set_option('display.max_columns', None)
```

In [2]:
```
#na_filter set to False as otherwise empty strings are interpreted as NaN
df_tweets_cleaned = pd.read_csv('..\data\Tweets_cleaned.csv', encoding='utf-8', na_filter= False)
```

Feature Engineering

Let's generate some features we could possibly use. Some features, such as `emojis_flag`, `emoticons_flag`, and `hashtags_flag` are already generated. Below are some of the features we are engineering/generating:

1. `emojis_num` denotes the number of emojis used in a tweet.
2. `emoitcons_num` denotes the number of emoticons used in a tweet.
3. `hashtag_num` denotes the number of hashtags used in a tweet.
4. `numbers_flag` denotes whether the tweet contains numbers or not (either in Arabic or English)
5. `numbers_num` denotes the number of times a tweet contains numbers We noticed that numbers were used in quite a few negative tweets, such as hours, time, dollars, flight numbers, etc. This is why we are generating a binary flag, as well as a numeric count of numbers used in a tweet.
6. `char_length_original` denotes the length of the the user's original tweet. This includes everything (@ mentions, RT retweets, hyperlinks, etc.)
7. `char_length_user` denotes the length of the user's cleaned tweet. The length will be based off the column `text_cleaned` We also noticed that negative tweets were, on average, longer than positive tweets in terms of character length.
8. `mentions_num` denotes the number of mentions a tweet has (@ mentions)
9. `retweet_flag` denotes if the user's tweet retweeted a tweet (normally the retweet is one of an airline, rarely another user). No need to create a count for retweets in a user's tweet because it's always 1.
10. `http_flag` denotes if the user's tweet has a HTTP link. No need to create a count for http links in a user's tweet because it's always 1 too.

The True/Flase `_flag` will need to be converted into binary flags instead (i.e. True/False into 1/0).

Any of the `_num` columns will likely need to be scaled to a scale from 0 to 1.

We will also need to vectorize the words in the tweets. To do so, there are several ways of doing so. We could use `word2vec`, `emoji2vec`, or a combination of both of them called `phrase2vec`.

Lastly, we will need to convert airline_sentiment into 0 or 1. In this situation, because we care about classifying negative sentiment tweets, and not really care about whether it's positive or neutral, we decided to group the positive and neutral tweets as `non-negative`. All `non-negative` tweets are class `0`, whereas all `negative` tweets are class `1`.

Generate columns `emojis_num`, `emoticons_num`, and `hashtag_num`

Generate basic features such as `emojis_num`, `emoticons_num`, `hashtag_num` from already developed columns.

In [3]:
```
#creates emojis_num column
def create_emojis_num(df):
    df['emojis_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'emojis_flag']:
            tweet_emojis = df.at[i, 'emojis']
            #strip brackets, quote, and spaces
            tweet_emojis_list = list(tweet_emojis.strip('[]').replace("\'", "").strip().split(","))
            emoji_counter = 0

            for emoji in tweet_emojis_list:
                emoji_counter = emoji_counter + 1

            df.at[i, 'emojis_num'] = emoji_counter
        else:
            df.at[i, 'emojis_num'] = 0

    return df

#creates emoticons_num column
def create_emoticons_num(df):
    df['emoticons_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'emoticons_flag']:
            tweet_emoticons = df.at[i, 'emoticons']
            #strip brackets, quote, and spaces
            tweet_emoticons_list = list(tweet_emoticons.strip('[]').replace("\'", "").strip().split(
","))
            emoticons_counter = 0

            for emoticon in tweet_emoticons_list:
                emoticons_counter = emoticons_counter + 1

            df.at[i, 'emoticons_num'] = emoticons_counter
        else:
            df.at[i, 'emoticons_num'] = 0

    return df

#creates hashtag_num column
def create_hashtags_num (df):
    df['hashtags_num'] = 0

    for i, row in df.iterrows():
        if df.at[i, 'hashtags_flag']:
            tweet_hashtags = df.at[i, 'hashtags']
            #strip brackets, quote, and spaces
            tweet_hashtags_list = list(tweet_hashtags.strip('[]').replace("\'", "").strip().split(","
))
            hashtags_counter = 0

            for hashtag in tweet_hashtags_list:
                hashtags_counter = hashtags_counter + 1

            df.at[i, 'hashtags_num'] = hashtags_counter
        else:
            df.at[i, 'hashtags_num'] = 0

    return df
```

In [4]:
```
df_tweets_cleaned = create_emojis_num(df_tweets_cleaned)
df_tweets_cleaned = create_emoticons_num(df_tweets_cleaned)
df_tweets_cleaned = create_hashtags_num(df_tweets_cleaned)
```

In [5]:
```
#df_tweets_cleaned.loc[df_tweets_cleaned['hashtags_flag'] == True]
```

Generate columns `numbers_flag`, `numbers_num`

Generate a binary flag and a count of how many times numbers were used in a tweet. Numbers can either be numeric, or in English. English numbers are sometimes considered stop words by Spacy (e.g. "twelve" is a stop word in tweet `568911315026063361`, but "thirty" is not for some reason in tweet `568237684277141504`), and were removed in `lemmas_list`, so we generate the numbers features from column `text_cleaned_no_abbreviations`. We will use Spacy model to help us determine which token are like numbers, using `like_num`.

In [6]:
```
df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 568911315026063361]
```

Out[6]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline
2767	568911315026063361	negative	1.0	Late Flight	1.0	United

In [7]:
```
#load spacy model
import spacy

nlp = spacy.load('en_core_web_md')
```

In [8]:
```
#this function will create the columns numbers_flag and numbers_num
def create_numbers_columns(df):
    df['numbers_flag'] = False
    df['numbers_num'] = 0

    for i, row in df.iterrows():
        if i % 1000 == 0:
            print('at row number: ' + str(i))

        text = df.at[i, 'text_cleaned_no_abbreviations']
        #print(type(text))

        like_num_count = 0

        #tokenize text into list of tokens
        #print(text)

        token_list = nlp(text)

        #iterate through our tokens and count the number of nums
        for token in token_list:
            #print(token)
            if token.like_num:
                like_num_count = like_num_count + 1

        #at the end, we set our new columns
        if like_num_count != 0:
            df.at[i, 'numbers_flag'] = True
            df.at[i, 'numbers_num'] = like_num_count

    return df
```

In [9]:
```
#Sanity check

create_numbers_columns(df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 568911315026063361])
#create_numbers_columns(df_tweets_cleaned.loc[df_tweets_cleaned['tweet_id'] == 570093964059156481])
```

Out[9]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline
2767	568911315026063361	negative	1.0	Late Flight	1.0	United

In [10]:
```
df_tweets_cleaned = create_numbers_columns(df_tweets_cleaned)
```

```
at row number: 0
at row number: 1000
at row number: 2000
at row number: 3000
at row number: 4000
at row number: 5000
at row number: 6000
at row number: 7000
at row number: 8000
at row number: 9000
at row number: 10000
at row number: 11000
at row number: 12000
at row number: 13000
at row number: 14000
```

In [11]:
```
#df_tweets_cleaned.loc[df_tweets_cleaned['numbers_flag'] == True]
```

Generate columns `char_length_original`, `char_length_user`

Generate columns with the number of characters in original tweet, and cleaned tweet from column `text_cleaned`.

In [12]:
```
#this function will create the columns numbers_flag and numbers_num
def create_char_length_columns(df):
    df['char_length_original'] = 0
    df['char_length_user'] = 0

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        cleaned_text = df.at[i, 'text_cleaned_no_abbreviations']

        df.at[i, 'char_length_original'] = len(text)
        df.at[i, 'char_length_user'] = len(cleaned_text)

    return df
```

In [13]:
```
df_tweets_cleaned = create_char_length_columns(df_tweets_cleaned)
```

Generate columns `mentions_num`, `retweet_flag`, and `http_flag`

Generate columns `mentions_num`: number of mentions in a tweet, `retweet_flag`: whether a tweet has a retweet, and `http_flag`: whether a tweet has a http link.

In [14]:
```
import re

#this function will create mentions_num column
def create_mentions_num(df):
    df['mentions_num'] = 0

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'\@[\w\d]*'

        regex_hits_list = re.findall(regex_to_find, text)
        df.at[i, 'mentions_num'] = len(regex_hits_list)

    return df

#this function will create retweet_flag column
def create_retweet_flag(df):
    df['retweet_flag'] = False

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'RT \@.*'

        regex_hits_list = re.findall(regex_to_find, text)
        if (len(regex_hits_list) != 0):
            df.at[i, 'retweet_flag'] = True

    return df

#this function will create http_flag column
def create_http_flag(df):
    df['http_flag'] = False

    for i, row in df.iterrows():
        text = df.at[i, 'text']
        regex_to_find = r'https*://[^\s]*'

        regex_hits_list = re.findall(regex_to_find, text)
        if (len(regex_hits_list) != 0):
            df.at[i, 'http_flag'] = True

    return df
```

In [15]:
```
df_tweets_cleaned = create_mentions_num(df_tweets_cleaned)
df_tweets_cleaned = create_retweet_flag(df_tweets_cleaned)
df_tweets_cleaned = create_http_flag(df_tweets_cleaned)
```

Scale numeric columns

The numeric columns will likely need to be scaled to a scale from 0 to 1. For columns `char_length_original` and `char_length_user` we will use normal MinMaxScaler because there aren't any big outliers, but for the other columns we will use RobustScaler as there are outliers.

In [16]:
```
df_tweets_cleaned[['emojis_num', 'emoticons_num', 'hashtags_num', 'numbers_num', 'char_length_original', 'char_length_user', 'mentions_num']].describe()
```

Out[16]:

	emojis_num	emoticons_num	hashtags_num	numbers_num	char_length_original	char_length_user	mentions_num
count	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000
mean	0.066667	0.019262	0.238525	0.429372	103.822063	88.186066	1.132719
std	0.612111	0.140400	0.654195	0.741321	36.277339	36.834301	0.410359
min	0.000000	0.000000	0.000000	0.000000	12.000000	0.000000	1.000000
25%	0.000000	0.000000	0.000000	0.000000	77.000000	59.000000	1.000000
50%	0.000000	0.000000	0.000000	0.000000	114.000000	96.000000	1.000000
75%	0.000000	0.000000	0.000000	1.000000	136.000000	121.000000	1.000000
max	40.000000	3.000000	8.000000	7.000000	186.000000	177.000000	6.000000

In [17]:
```
from sklearn.preprocessing import MinMaxScaler, RobustScaler

minMaxScaler = MinMaxScaler()
df_tweets_cleaned['char_length_original_scaled'] = 0
df_tweets_cleaned['char_length_user_scaled'] = 0
df_tweets_cleaned[['char_length_original_scaled', 'char_length_user_scaled']] = \
    minMaxScaler.fit_transform(df_tweets_cleaned[['char_length_original', 'char_length_user']])

robustScaler = RobustScaler()
df_tweets_cleaned['emojis_num_scaled'] = 0
df_tweets_cleaned['emoticons_num_scaled'] = 0
df_tweets_cleaned['hashtags_num_scaled'] = 0
df_tweets_cleaned['numbers_num_scaled'] = 0
df_tweets_cleaned['mentions_num_scaled'] = 0
df_tweets_cleaned[['emojis_num_scaled', 'emoticons_num_scaled', 'hashtags_num_scaled', 'numbers_num_scaled', 'mentions_num_scaled']] = \
    minMaxScaler.fit_transform(df_tweets_cleaned[['emojis_num', 'emoticons_num', 'hashtags_num', 'numbers_num', 'mentions_num']])
```

Convert binary True/False columns to 1s/0s

We should convert binary True/False columns to 1s/0s.

In [18]:
```
df_tweets_cleaned['emojis_flag'] = df_tweets_cleaned['emojis_flag'].astype(int)
df_tweets_cleaned['emoticons_flag'] = df_tweets_cleaned['emoticons_flag'].astype(int)
df_tweets_cleaned['hashtags_flag'] = df_tweets_cleaned['hashtags_flag'].astype(int)
df_tweets_cleaned['numbers_flag'] = df_tweets_cleaned['numbers_flag'].astype(int)
df_tweets_cleaned['retweet_flag'] = df_tweets_cleaned['retweet_flag'].astype(int)
df_tweets_cleaned['http_flag'] = df_tweets_cleaned['http_flag'].astype(int)
```

Group the positive and neutral

As stated before, our goal is to predict negative sentiment tweets; we don't particularly care if the tweets are positive or neutral, to us they are the same thing: not negative. Therefore, we merge the positive and neutral classes to 0s, and rename negative label to 1s.

In [19]:
```
df_tweets_cleaned['binary_response_variable'] = False

df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'neutral', 'binary_response_variable'] = False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'positive', 'binary_response_variable'] = False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'negative', 'binary_response_variable'] = True
```

In [20]:
```
df_tweets_cleaned.binary_response_variable
```

Out[20]:
```
0        False
1        False
2        False
3         True
4         True
         ...
14635    False
14636     True
14637    False
14638     True
14639    False
Name: binary_response_variable, Length: 14640, dtype: bool
```

In [21]:
```
df_tweets_cleaned.binary_response_variable = df_tweets_cleaned.binary_response_variable.astype(int)
```

In [22]:
```
df_tweets_cleaned
```

Out[22]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airlir
0	570306133677760513	neutral	1.0000			Virg Americ
1	570301130888122368	positive	0.3486		0.0	Virg Americ
2	570301083672813571	neutral	0.6837			Virg Americ
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virg Americ
4	570300817074462722	negative	1.0000	Can't Tell	1.0	Virg Americ
...	...	...	...	...	...	
14635	569587686496825344	positive	0.3487		0.0	America
14636	569587371693355008	negative	1.0000	Customer Service Issue	1.0	America
14637	569587242672398336	neutral	1.0000			America
14638	569587188687634433	negative	1.0000	Customer Service Issue	0.6659	America
14639	569587140490866689	neutral	0.6771		0.0	America

14640 rows × 39 columns

Get rid of all other columns

We only need the `binary_response_variable`, `_flag` columns, `_scaled` columns, and `lemmas_list` column (this will be vectorized using our models).

In [23]:
```
df_tweets_cleaned.columns
```

Out[23]:
```
Index(['tweet_id', 'airline_sentiment', 'airline_sentiment_confidence',
       'negativereason', 'negativereason_confidence', 'airline', 'text',
       'text_cleaned', 'text_cleaned_time_removed', 'emojis_flag', 'emojis',
       'emoticons_flag', 'emoticons', 'text_cleaned_without_emojis_emoticons',
       'hashtags', 'text_cleaned_without_emojis_emoticons_hashtags',
       'hashtags_flag', 'text_cleaned_lower_case',
       'text_cleaned_no_abbreviations', 'text_list_no_stop_words',
       'lemmas_list', 'emojis_num', 'emoticons_num', 'hashtags_num',
       'numbers_flag', 'numbers_num', 'char_length_original',
       'char_length_user', 'mentions_num', 'retweet_flag', 'http_flag',
       'char_length_original_scaled', 'char_length_user_scaled',
       'emojis_num_scaled', 'emoticons_num_scaled', 'hashtags_num_scaled',
       'numbers_num_scaled', 'mentions_num_scaled',
       'binary_response_variable'],
      dtype='object')
```

In [24]:
```
df_tweets_cleaned = df_tweets_cleaned[[
    'binary_response_variable',
    'emojis_flag',
    'emoticons_flag',
    'hashtags_flag',
    'numbers_flag',
    'retweet_flag',
    'http_flag',
    'char_length_original_scaled',
    'char_length_user_scaled',
    'emojis_num_scaled',
    'emoticons_num_scaled',
    'hashtags_num_scaled',
    'numbers_num_scaled',
    'mentions_num_scaled',
    'lemmas_list'
]]
```

Baseline Benchmark with trained word2vec gensim model

We do a baseline benchmark with Patrick's trained word2vec gensim model, located at `..\milestone1\Patrick\gensim_word2vec_trained.bin` of this repo.

Split into train and test datasets

We need to split our dataframe into train and test datasets/dataframes. We don't need a validation set for now, but we will split into one later for our machine learning model tuning. For now, 70%/30% for train/test seems good enough. We will split based on column `binary_response_variable`.

In [25]:
```
from sklearn.model_selection import train_test_split

X = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns != 'binary_response_variable']
Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_variable']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1, stratify=Y)
```

Generate Tweet vectors

In [26]:
```
import gensim

custom_w2v_gensim_model_path = '..\milestone1\Patrick\gensim_word2vec_trained.bin'
w2v_gensim_model = gensim.models.KeyedVectors.load_word2vec_format(custom_w2v_gensim_model_path, binar
y=True)
```

In [27]:
```
#calculates a vector for a given Tweet
def calculate_average_tweet_vector(tweet, w2v_model, num_dimensions):
    tokens = tweet.split(' ')

    tweet_vector = np.zeros(num_dimensions, np.float32)
    actual_token_count = 0

    for token in  tokens:
        if token in w2v_model.wv.vocab:
            actual_token_count = actual_token_count + 1
            tweet_vector = np.add(tweet_vector, w2v_model[token])

    tweet_vector = np.divide(tweet_vector, actual_token_count)

    return tweet_vector
```

In [28]:
```
#Sanity check
#calculate_average_tweet_vector("arrangement reimburse rental", w2v_gensim_model, w2v_gensim_model.wv.
vector_size)
```

In [29]:
```
num_dimensions = w2v_gensim_model.wv.vector_size
X_train['tweet_vector'] = X_train.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text,
w2v_gensim_model, num_dimensions))
X_test['tweet_vector'] = X_test.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text, w2
v_gensim_model, num_dimensions))
```

In [30]:
```
df_train_tweet_vector = pd.DataFrame(list(X_train['tweet_vector']), index=X_train.index)
df_test_tweet_vector = pd.DataFrame(list(X_test['tweet_vector']), index=X_test.index)
```

In [31]:
```
#after creating new dataframes above, any zeroes are converted to null...we need to convert them back
 to zeroes...
df_train_tweet_vector = df_train_tweet_vector.fillna(0)
df_test_tweet_vector = df_test_tweet_vector.fillna(0)
```

In [32]:
```
X_train_combined = pd.concat([df_train_tweet_vector, X_train], axis=1)
X_test_combined = pd.concat([df_test_tweet_vector, X_test], axis=1)
```

Remove `tweet_vector` and `lemmas_list` column

In [33]:
```
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'tweet_vector']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'tweet_vector']
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'lemmas_list']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'lemmas_list']
```

In [34]:
```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(C=100)
logmodel.fit(X_train_combined, Y_train)
predictions = logmodel.predict(X_test_combined)
```

In [35]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.71      0.75      1639
           1       0.84      0.89      0.86      2753

   micro avg       0.82      0.82      0.82      4392
   macro avg       0.81      0.80      0.81      4392
weighted avg       0.82      0.82      0.82      4392
```

Baseline Benchmark with Google word2vec gensim model subset

We do a baseline benchmark with Google's word2vec gensim model that is subset with our word vectors as well, located at `..\milestone1\Patrick\gensim_word2vec_google_subset.bin` , which can be unzipped from `..\milestone1\Patrick\gensim_word2vec_google_subset.bin` of this repo.

Split into train and test datasets

We need to split our dataframe into train and test datasets/dataframes. We don't need a validation set for now, but we will split into one later for our machine learning model tuning. For now, 70%/30% for train/test seems good enough. We will split based on column `binary_response_variable` .

In [36]:
```
from sklearn.model_selection import train_test_split

X = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns != 'binary_response_variable']
Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_variable']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1, stratify=Y)
```

Generate Tweet vectors

In [37]:
```
import gensim

custom_w2v_gensim_model_path = '..\milestone1\Patrick\gensim_word2vec_google_subset.bin'
w2v_gensim_model = gensim.models.KeyedVectors.load_word2vec_format(custom_w2v_gensim_model_path, binar
y=True)
```

In [38]:
```
#calculates a vector for a given Tweet
def calculate_average_tweet_vector(tweet, w2v_model, num_dimensions):
    tokens = tweet.split(' ')

    tweet_vector = np.zeros(num_dimensions, np.float32)
    actual_token_count = 0

    for token in  tokens:
        if token in w2v_model.wv.vocab:
            actual_token_count = actual_token_count + 1
            tweet_vector = np.add(tweet_vector, w2v_model[token])

    tweet_vector = np.divide(tweet_vector, actual_token_count)

    return tweet_vector
```

In [39]:
```
#Sanity check
#calculate_average_tweet_vector("arrangement reimburse rental", w2v_gensim_model, w2v_gensim_model.wv.
vector_size)
```

In [40]:
```
num_dimensions = w2v_gensim_model.wv.vector_size
X_train['tweet_vector'] = X_train.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text,
w2v_gensim_model, num_dimensions))
X_test['tweet_vector'] = X_test.lemmas_list.apply(lambda text: calculate_average_tweet_vector(text, w2
v_gensim_model, num_dimensions))
```

In [41]:
```
df_train_tweet_vector = pd.DataFrame(list(X_train['tweet_vector']), index=X_train.index)
df_test_tweet_vector = pd.DataFrame(list(X_test['tweet_vector']), index=X_test.index)
```

In [42]:
```
#after creating new dataframes above, any zeroes are converted to null...we need to convert them back
 to zeroes...
df_train_tweet_vector = df_train_tweet_vector.fillna(0)
df_test_tweet_vector = df_test_tweet_vector.fillna(0)
```

In [43]:
```
X_train_combined = pd.concat([df_train_tweet_vector, X_train], axis=1)
X_test_combined = pd.concat([df_test_tweet_vector, X_test], axis=1)
```

Remove tweet_vector and lemmas_list column

In [44]:
```
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'tweet_vector']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'tweet_vector']
X_train_combined = X_train_combined.loc[:, X_train_combined.columns != 'lemmas_list']
X_test_combined = X_test_combined.loc[:, X_test_combined.columns != 'lemmas_list']
```

In [45]:
```
from sklearn.linear_model import LogisticRegression

logmodel = LogisticRegression(C=100)
logmodel.fit(X_train_combined, Y_train)
predictions = logmodel.predict(X_test_combined)
```

In [46]:
```
from sklearn.metrics import classification_report
print(classification_report(Y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.79      0.70      0.74      1639
           1       0.83      0.89      0.86      2753

   micro avg       0.82      0.82      0.82      4392
   macro avg       0.81      0.80      0.80      4392
weighted avg       0.82      0.82      0.82      4392
```

In [47]:
```
predictions
```

Out[47]:
```
array([1, 1, 1, ..., 0, 1, 0])
```

In [ ]:

In [1]:
```
#import warnings
#warnings.filterwarnings('ignore')

#sklearn models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, \
        GradientBoostingClassifier, StackingClassifier, BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier

#sklearn metrics
from sklearn.metrics import roc_curve, auc, average_precision_score, precision_recall_curve
#DOES NOT WORK: , plot_precision_recall_curve
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_score, f1_score, roc_auc_
score

#sklearn model selection
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split, cross_val_predi
ct

#scikit-plot
from scikitplot.metrics import plot_precision_recall, plot_roc, plot_confusion_matrix
from scikitplot.estimators import plot_learning_curve

#mlxtend
from mlxtend.plotting import plot_learning_curves, plot_decision_regions


#hyperparameter tuning

#miscellaneous
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#some "global" variables defined here
random_state_num = 0
cv_num = 10
classes_num = 2 #there are two classes, non-negative (0) and negative (1)
fig_size_tuple = (15,7)
title_fontsize_num = 15
label_fontsize_num = 12
```

Read Trained Word2Vec Embedding Vectors and Engineered Features

We read in the engineered features dataset as a dataframe.

In [2]:
```
#read the csv of engineered features
#then split the columns into
df_gensim_word2vec_features = pd.read_csv('..\data\gensim_word2vec_trained_with_engineered_features.cs
v')
X = df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns != 'binary_response_variabl
e']
Y= df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns == 'binary_response_variabl
e']
```

Split into Train and Test dataset

The dataset is then split into a train and test dataset.

In [3]:
```
#Split into training set, and test set
# we do not need a validation set because we will be doing k-fold cross validation

#split into 0.7 training, and 0.3 testing set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = random_state_n
um)

#convert Y_train and Y_test into 1-d array so we don't get stupid warnings about needing a 1d array fo
r Y, in cross_validate function
Y_train_1d = Y_train.values.ravel()
Y_test_1d = Y_test.values.ravel()

#convert Y_train and Y_test into numpy array
Y_train_np_array = Y_train.to_numpy()
Y_test_np_array = Y_test.to_numpy()
```

Models

Here we define a set of "basic" models we use as benchmarks to see which model algorithm is better than the others.

In [4]:
```
stacking_base_learners = [
    ('sbl_1', LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, \
                                                 class_weight={1: 0.4, 0: 0.6}, penalty='l1', solver='lib
linear', )),
    ('sbl_2', KNeighborsClassifier(n_neighbors=3, )),
    ('sbl_3', DecisionTreeClassifier()),
    ('sbl_4', GaussianNB())
]

models = [
# Hyperparameter tuned GBM model
#     GradientBoostingClassifier(random_state = random_state_num, min_samples_split=140, min_samples_l
eaf=14, \
#                                subsample=0.9, n_estimators=10, max_features='sqrt', max_depth=8, \
#                                loss='deviance', learning_rate=0.15000000000000002, criterion='friedm
an_mse')

    KNeighborsClassifier(n_neighbors=3, ),
    DecisionTreeClassifier(),
    #SVC(C = 1000000, gamma = 'auto', kernel = 'rbf', probability=True),
    GaussianNB(),
    #increased max_iter because it failed to converge at 100
    LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, class_weight={1: 0.4, 0:
0.6}, \
                       penalty='l1', solver='liblinear'),
    AdaBoostClassifier(random_state = random_state_num),
    RandomForestClassifier(random_state = random_state_num),
    GradientBoostingClassifier(random_state = random_state_num),
    BaggingClassifier(base_estimator=LogisticRegression(random_state = random_state_num, max_iter = 50
0, C=1, \
                                                        class_weight={1: 0.4, 0: 0.6}, penalty='l1', s
olver='liblinear')),
    StackingClassifier(estimators=stacking_base_learners, final_estimator=LogisticRegression())
]
```

Hyperparameter Tuning

Hyperparameter tuning is the adjustment of various pre-execution parameters passed to our Machine Learning models that affect their training/execution. Here we use two automated methods of choosing from a wide set of specified parameters - Grid Search (which exhaustively tries all combinations of specified parameters) and Random Search (which tries randomly sampled combinations of parameters).

Grid Search

Grid Search (exhaustive) hyperparameter tuning.

In [5]:
```
#%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

from sklearn.model_selection import GridSearchCV

for model in models:
    model_name = model.__class__.__name__

    #----Logistic Regression Hyperparameter Tuning----

    if model_name == 'LogisticRegression':

        penalty = ['l1', 'l2']
        C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
        class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
        solver = ['liblinear', 'saga']

        param_grid = dict(penalty=penalty,
                          C=C,
                          class_weight=class_weight,
                          solver=solver)

        grid = GridSearchCV(estimator=model,
                            param_grid=param_grid,
                            scoring='roc_auc',
                            verbose=1,
                            n_jobs=-1)
        grid_result = grid.fit(X_train, Y_train)
        print('Model Name: ', model_name)
        print('Best Score: ', grid_result.best_score_)
        print('Best Params: ', grid_result.best_params_)

        #----Gradient Boosting Hyperparameter Tuning----

#     if model_name == 'GradientBoostingClassifier':

#         learning_rate = [0.15,0.1,0.05,0.01,0.005,0.001]
#         n_estimators = [100,250,500,750,1000,1250,1500,1750]
#         max_depth = [2,3,4,5,6,7]

#         param_grid = dict(learning_rate=learning_rate,
#                           n_estimators=n_estimators,
#                           max_depth=max_depth,)

#         grid = GridSearchCV(estimator=model,
#                             param_grid=param_grid,
#                             scoring='roc_auc',
#                             verbose=1,
#                             n_jobs=-1)
#         grid_result = grid.fit(X_train, Y_train)
#         print('Model Name: ', model_name)
#         print('Best Score: ', grid_result.best_score_)
#         print('Best Params: ', grid_result.best_params_)
```

```
Fitting 5 folds for each of 128 candidates, totalling 640 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:    2.8s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:   44.3s
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 640 out of 640 | elapsed:  5.3min finished
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\sklearn\utils\validation.py:7
60: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Model Name:  LogisticRegression
Best Score:  0.883580552276247
Best Params:  {'C': 1, 'class_weight': {1: 0.4, 0: 0.6}, 'penalty': 'l1', 'solver': 'liblinear'}
```

Random Search

Random Search hyperparameter tuning.

In [6]:
```
#%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

#exploration of specifying a range of values using numpy's logspace function.
#Used to specify range of C values in logistic regression.

import numpy as np

np.set_printoptions(precision=10)
np.set_printoptions(suppress=True)

np.logspace(0, 4, num=10)
```

Out[6]:
```
array([    1.          ,     2.7825594022,     7.7426368268,
          21.5443469003,    59.9484250319,   166.81005372  ,
         464.1588833613,  1291.5496650149,  3593.8136638046,
       10000.          ])
```

In [7]:

```
#%%script false --no-raise-error
# ^ disables this cell in jupyter notebook

from sklearn.model_selection import RandomizedSearchCV

for model in models:
    model_name = model.__class__.__name__

    print(model)
    if model_name == 'LogisticRegression':

        penalty = ['l1', 'l2']
        C = np.logspace(0, 4, num=10)
        class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
        solver = ['liblinear', 'saga']

        param_distributions = dict(penalty=penalty,
                          C=C,
                          class_weight=class_weight,
                          solver=solver)

        random = RandomizedSearchCV(estimator=model,
                                    param_distributions=param_distributions,
                                    scoring='roc_auc',
                                    verbose=1, n_jobs=-1,
                                    n_iter=100)
        random_result = random.fit(X_train, Y_train)

        print('Model Name: ', model_name)
        print('Best Score: ', random_result.best_score_)
        print('Best Params: ', random_result.best_params_)

    if model_name == 'GradientBoostingClassifier':

        loss=["deviance"]
        learning_rate=np.linspace(0.05, 0.2, num=4)
        max_depth=[3,5,8]
        max_features=["log2","sqrt"]
        criterion=["friedman_mse",  "mae"]
        subsample=[0.5, 0.618, 0.8, 0.85, 0.9, 0.95, 1.0]
        n_estimators=[10]

        param_distributions = dict(loss=loss,
                          learning_rate=learning_rate,
                        #  min_samples_split=min_samples_split,
                        #  min_samples_leaf=min_samples_leaf,
                          max_depth=max_depth,
                          max_features=max_features,
                          criterion=criterion,
                          subsample=subsample,
                          n_estimators=n_estimators)

        random = RandomizedSearchCV(estimator=model,
                                    param_distributions=param_distributions,
                                    cv=3,
                                    scoring='roc_auc',
                                    verbose=1,
                                    n_jobs=-1,
                                    n_iter=100)
        random_result = random.fit(X_train, Y_train)

        print('Model Name: ', model_name)
        print('Best Score: ', random_result.best_score_)
        print('Best Params: ', random_result.best_params_)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
GaussianNB(priors=None, var_smoothing=1e-09)
LogisticRegression(C=1, class_weight={0: 0.6, 1: 0.4}, dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=500, multi_class='auto', n_jobs=None, penalty='l1',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
Fitting 5 folds for each of 100 candidates, totalling 500 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:   10.6s
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 426 tasks      | elapsed:  4.6min
[Parallel(n_jobs=-1)]: Done 500 out of 500 | elapsed:  5.4min finished
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\sklearn\utils\validation.py:7
60: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change t
he shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

Model Name:  LogisticRegression
Best Score:  0.883580552276247
Best Params:  {'solver': 'liblinear', 'penalty': 'l1', 'class_weight': {1: 0.4, 0: 0.6}, 'C': 1.0}
AdaBoostClassifier(algorithm='SAMME.R', base_estimator=None, learning_rate=1.0,
                   n_estimators=50, random_state=0)
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
GradientBoostingClassifier(ccp_alpha=0.0, criterion='friedman_mse', init=None,
                           learning_rate=0.1, loss='deviance', max_depth=3,
                           max_features=None, max_leaf_nodes=None,
                           min_impurity_decrease=0.0, min_impurity_split=None,
                           min_samples_leaf=1, min_samples_split=2,
                           min_weight_fraction_leaf=0.0, n_estimators=100,
                           n_iter_no_change=None, presort='deprecated',
                           random_state=0, subsample=1.0, tol=0.0001,
                           validation_fraction=0.1, verbose=0,
                           warm_start=False)
Fitting 3 folds for each of 100 candidates, totalling 300 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done  26 tasks      | elapsed:  1.1min
[Parallel(n_jobs=-1)]: Done 176 tasks      | elapsed:  6.6min
[Parallel(n_jobs=-1)]: Done 300 out of 300 | elapsed:  9.8min finished
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\sklearn\ensemble\_gb.py:1454:
DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the s
hape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

```
Model Name:  GradientBoostingClassifier
Best Score:  0.8574078131617336
Best Params:  {'subsample': 1.0, 'n_estimators': 10, 'max_features': 'sqrt', 'max_depth': 8, 'loss':
'deviance', 'learning_rate': 0.15000000000000002, 'criterion': 'friedman_mse'}
BaggingClassifier(base_estimator=LogisticRegression(C=1,
                                                    class_weight={0: 0.6,
                                                                  1: 0.4},
                                                    dual=False,
                                                    fit_intercept=True,
                                                    intercept_scaling=1,
                                                    l1_ratio=None, max_iter=500,
                                                    multi_class='auto',
                                                    n_jobs=None, penalty='l1',
                                                    random_state=0,
                                                    solver='liblinear',
                                                    tol=0.0001, verbose=0,
                                                    warm_start=False),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=1.0, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
StackingClassifier(cv=None,
                   estimators=[('sbl_1',
                                LogisticRegression(C=1,
                                                   class_weight={0: 0.6,
                                                                 1: 0.4},
                                                   dual=False,
                                                   fit_intercept=True,
                                                   intercept_scaling=1,
                                                   l1_ratio=None, max_iter=500,
                                                   multi_class='auto',
                                                   n_jobs=None, penalty='l1',
                                                   random_state=0,
                                                   solver='liblinear',
                                                   tol=0.0001, verbose=0,
                                                   warm_start=False)),
                               ('sbl_2',
                                KNeighborsClassifier(algorithm='auto',
                                                     leaf_size=...
                                GaussianNB(priors=None, var_smoothing=1e-09))],
                   final_estimator=LogisticRegression(C=1.0, class_weight=None,
                                                      dual=False,
                                                      fit_intercept=True,
                                                      intercept_scaling=1,
                                                      l1_ratio=None,
                                                      max_iter=100,
                                                      multi_class='auto',
                                                      n_jobs=None, penalty='l2',
                                                      random_state=None,
                                                      solver='lbfgs',
                                                      tol=0.0001, verbose=0,
                                                      warm_start=False),
                   n_jobs=None, passthrough=False, stack_method='auto',
                   verbose=0)
```

Model Benchmarks using various metrics

Here we use different metrics to becnhmark the models we have defined above. Such metrics include balanced accuracy, precision, recall, F1, F2, and ROC-AUC.

F2 is different than F1 because it places heavier emphasis on recall, rather than precision. This is useful in our case because according to our business problem, there is more value to be gained from correctly identifying tweets of negative sentiment (i.e. the positive cases, or "1s"). Therefore, the opposite holds true as well: there is a heavier cost from misclassifying tweets of negative sentiment as positive sentiment (i.e. we misclassify true positives as false negatives); in real life, this could mean a potential PR disaster. On the other hand, misclassifying tweets of positive sentiment as negative sentiment isn't as costly (i.e. we misclassify true negatives as false positives); in real life, this just means the HR department may have to look at more false positives than a ML algorithm that places equal emphasis on both precision and recall.

In [8]:
```
#custom f2 score
#this places higher value on recall than precision
#i.e. a false negative has higher cost than a false positive
#(this makes sense: a negative tweet we don't catch classify us more than a positive tweet we don't cl
assify)
f2_scorer = make_scorer(fbeta_score, beta=2)

#for the other scorers, we need to create scorers from scratch since we want to use a dictionary for f
2_scorer under 'scoring'
balanced_accuracy_scorer = make_scorer(balanced_accuracy_score)
precision_scorer = make_scorer(precision_score)
recall_scorer = make_scorer(recall_score)
f1_scorer = make_scorer(f1_score)
roc_auc_scorer = make_scorer(roc_auc_score)
```

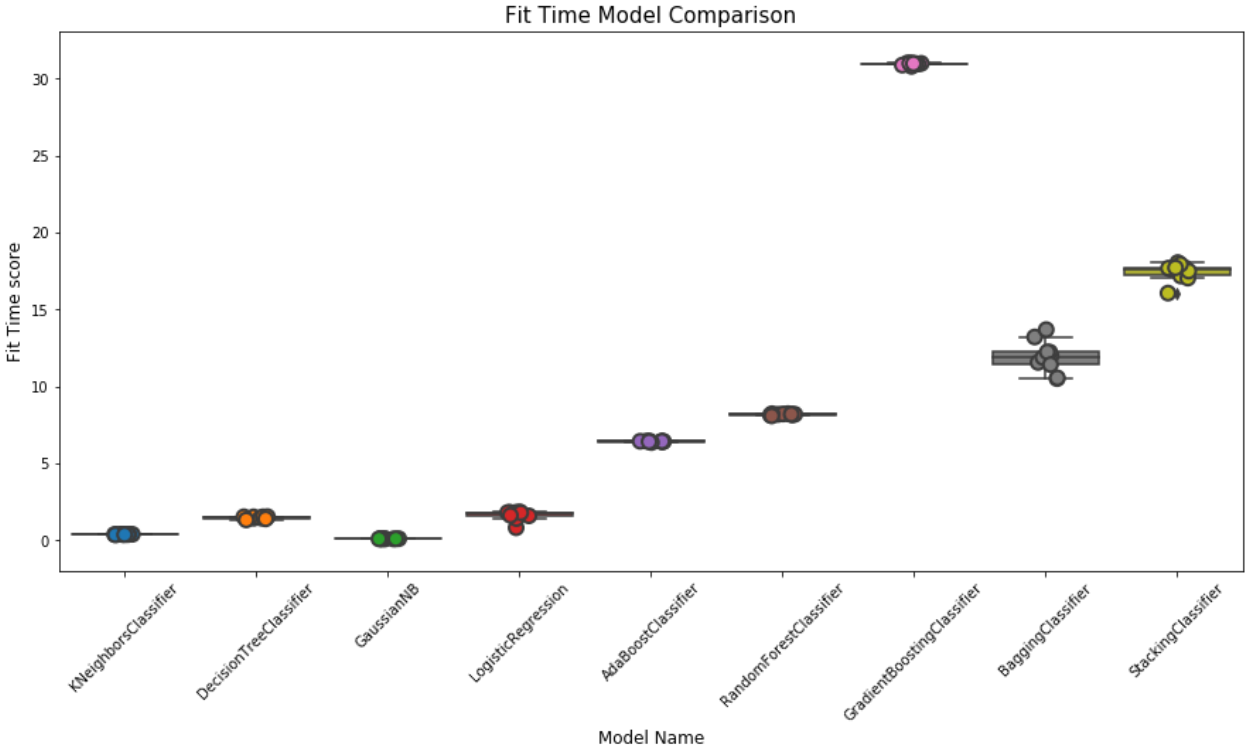
In [9]:
```
cv_result_entries = []

for model in models:
    model_name = model.__class__.__name__


    metrics_dict = cross_validate(
        model,
        X_train,
        Y_train_1d,
        #scoring = ['balanced_accuracy', 'precision', 'recall', 'f1', 'roc_auc'],
        scoring = {
            'balanced_accuracy': balanced_accuracy_scorer,
            'precision': precision_scorer,
            'recall': recall_scorer,
            'f1': f1_scorer,
            'f2': f2_scorer,
            'roc_auc': roc_auc_scorer
        },
        cv = cv_num,
        n_jobs=-1
    )

#REMOVED grid search hyperparam tuning code
#     penalty = ['l1', 'l2']
#     C = [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000]
#     class_weight = [{1:0.5, 0:0.5}, {1:0.4, 0:0.6}, {1:0.6, 0:0.4}, {1:0.7, 0:0.3}]
#     solver = ['liblinear', 'saga']

#     param_grid = dict(penalty=penalty,
#                       C=C,
#                       class_weight=class_weight,
#                       solver=solver)

#     grid = GridSearchCV(estimator=model,
#                         param_grid=param_grid,
#                         scoring='roc_auc',
#                         verbose=1,
#                         n_jobs=-1)
#     grid_result = grid.fit(X_train, Y_train)

#     print('Best Score: ', grid_result.best_score_)
#     print('Best Params: ', grid_result.best_params_)

    for metric_key in metrics_dict.keys():
        for fold_index, metric_score in enumerate(metrics_dict[metric_key]):
            cv_result_entries.append((model_name, fold_index, metric_key, metric_score))

#convert entries into a dataframe
df_cross_validate_results = pd.DataFrame(cv_result_entries, columns =['model_name', 'fold_index', 'met
ric_key', 'metric_score'])
```

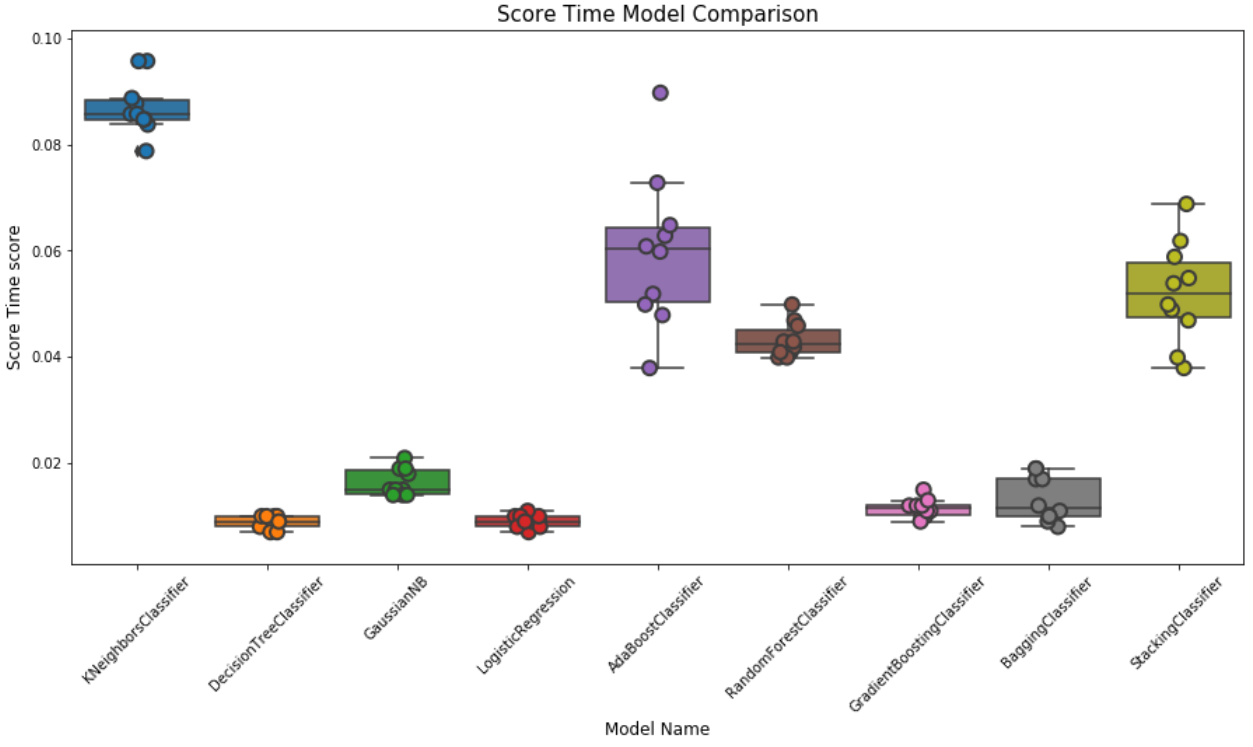
In [10]:
```
#save all results to CSV for reference
df_cross_validate_results.to_csv('../data/df_cross_validate_results.csv')
```

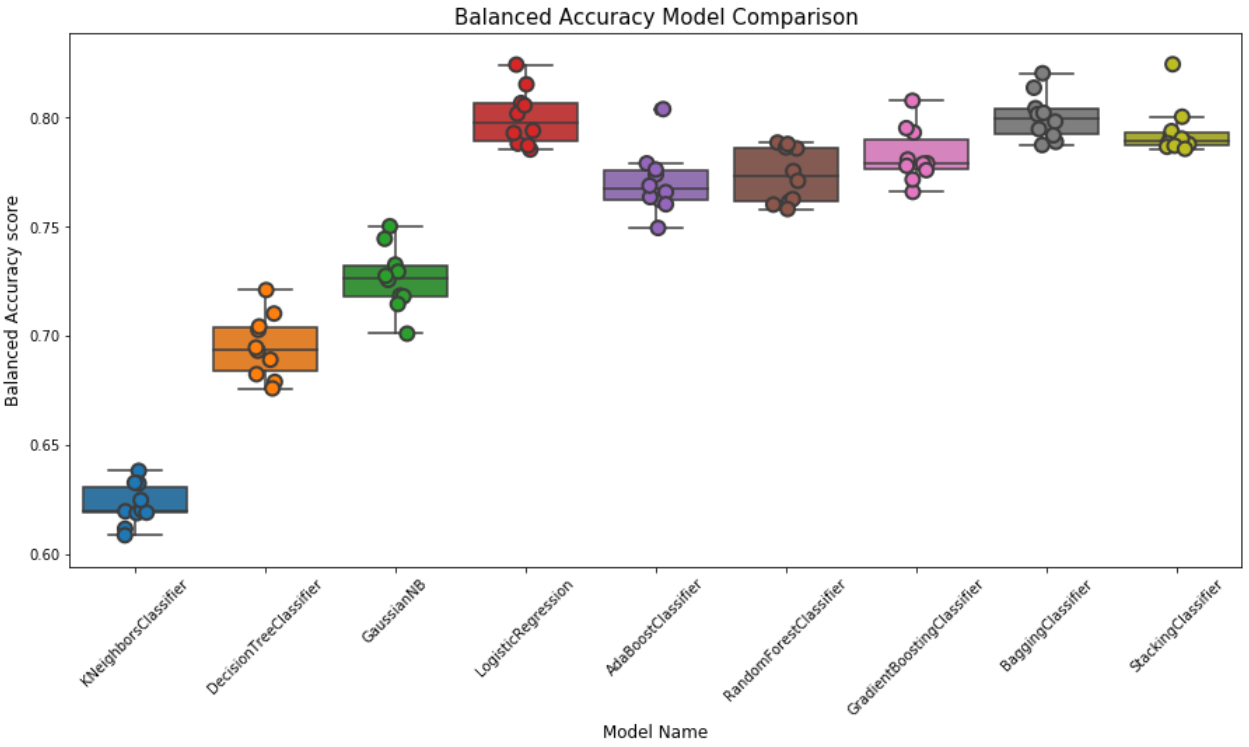
Plotting

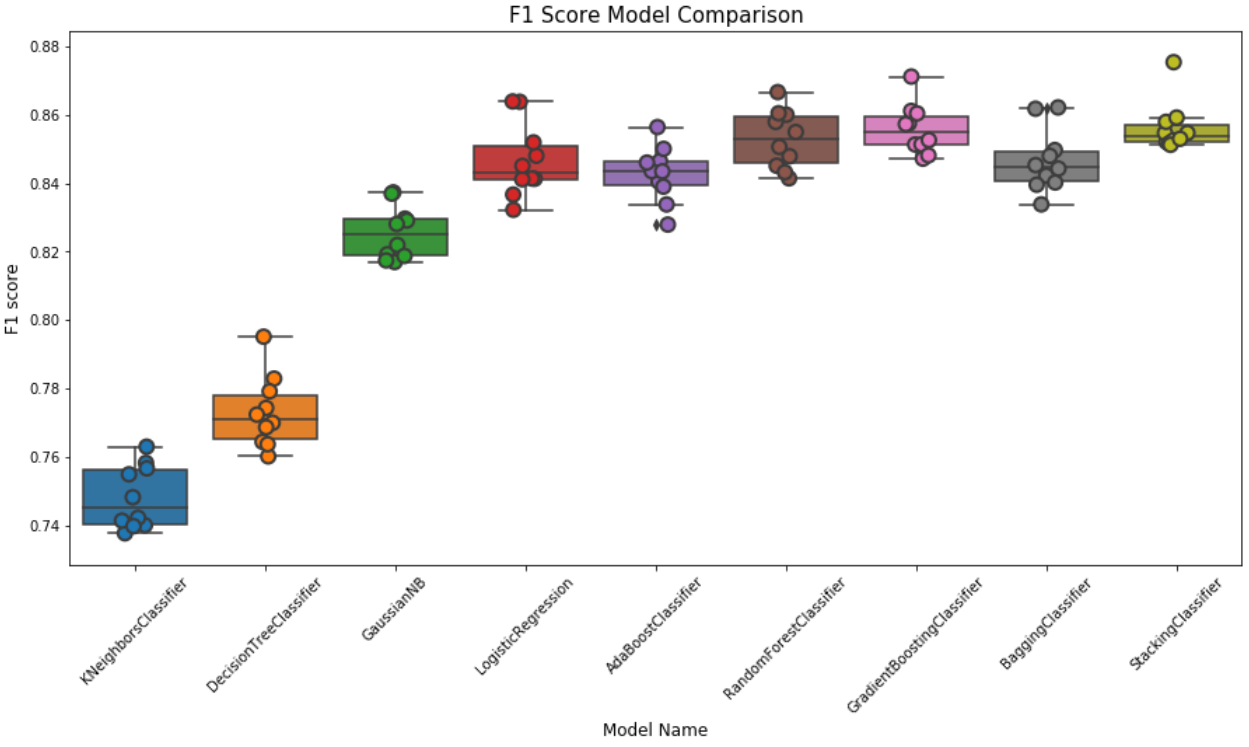
In [11]:
```
df_cv_results_fit_time = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'fit_time']
df_cv_results_score_time = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'score_time']
df_cv_results_balanced_acc = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_balanced_accuracy']
df_cv_results_precision = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_precision']
df_cv_results_recall = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_recall']
df_cv_results_f1 = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_f1']
df_cv_results_f2 = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_f2']
df_cv_results_roc_auc = df_cross_validate_results.loc[df_cross_validate_results.metric_key == 'test_roc_auc']


plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_fit_time)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_fit_time, size=10, linewidth=2)
plt.title('Fit Time Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('Fit Time score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_score_time)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_score_time, size=10, linewidth=2)
plt.title('Score Time Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('Score Time score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_balanced_acc)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_balanced_acc, size=10, linewidth=2)
plt.title('Balanced Accuracy Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('Balanced Accuracy score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_f1)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_f1, size=10, linewidth=2)
plt.title('F1 Score Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('F1 score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_f2)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_f2, size=10, linewidth=2)
plt.title('F2 Score Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('F2 score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_precision)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_precision, size=10, linewidth=2)
plt.title('Precision Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('Precision score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()
```

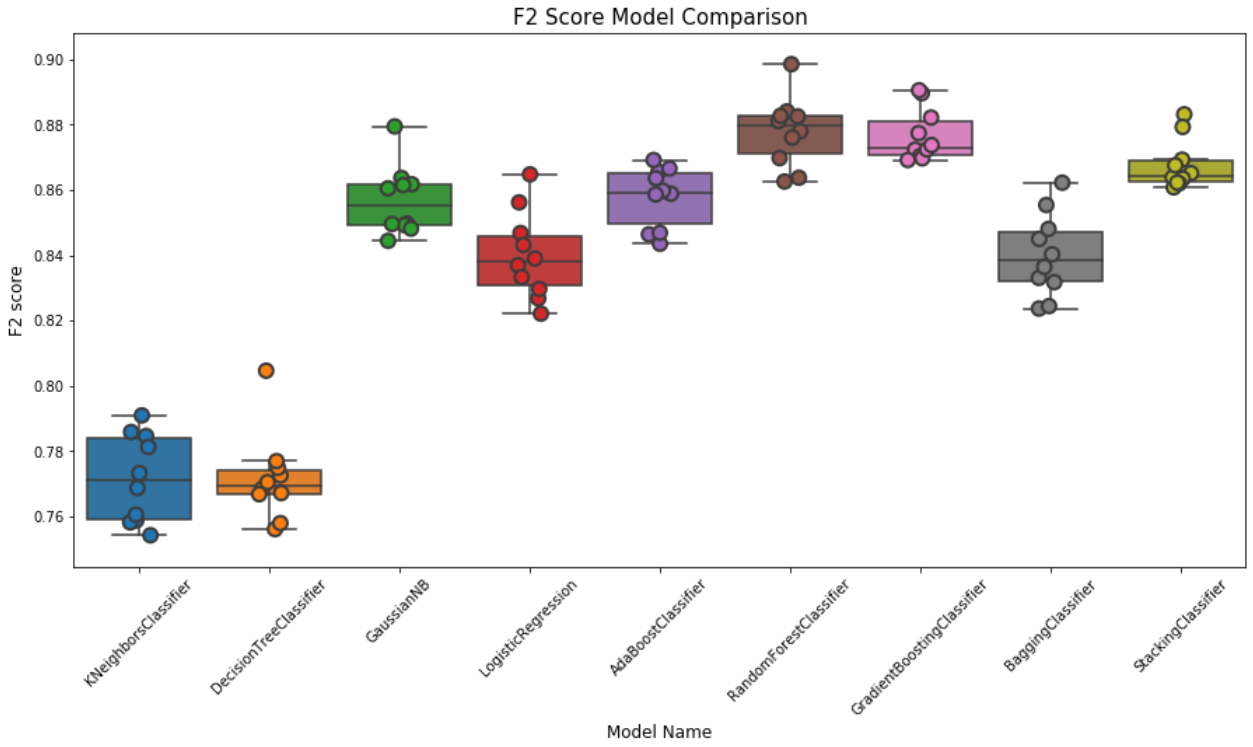
```
plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_recall)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_recall, size=10, linewidth=2)
plt.title('Recall Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('Recall score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=fig_size_tuple)
sns.boxplot(x='model_name', y='metric_score', data = df_cv_results_roc_auc)
sns.stripplot(x='model_name', y='metric_score', data = df_cv_results_roc_auc, size=10, linewidth=2)
plt.title('ROC-AUC Score Model Comparison', fontsize=title_fontsize_num)
plt.xlabel('Model Name', fontsize=label_fontsize_num)
plt.ylabel('ROC-AUC score', fontsize=label_fontsize_num)
plt.xticks(rotation=45)
plt.show()
```

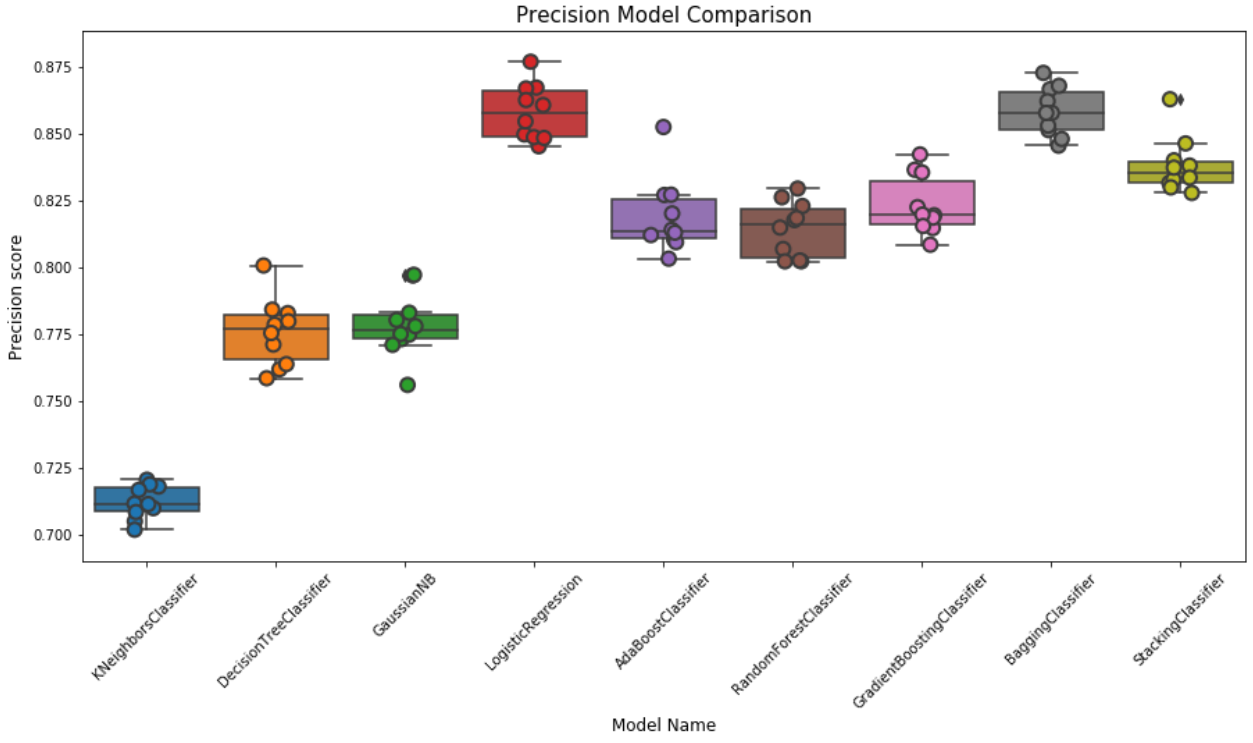

Fit Time Model Comparison
Fit Time score
30
25
20
15
10
5
0
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name

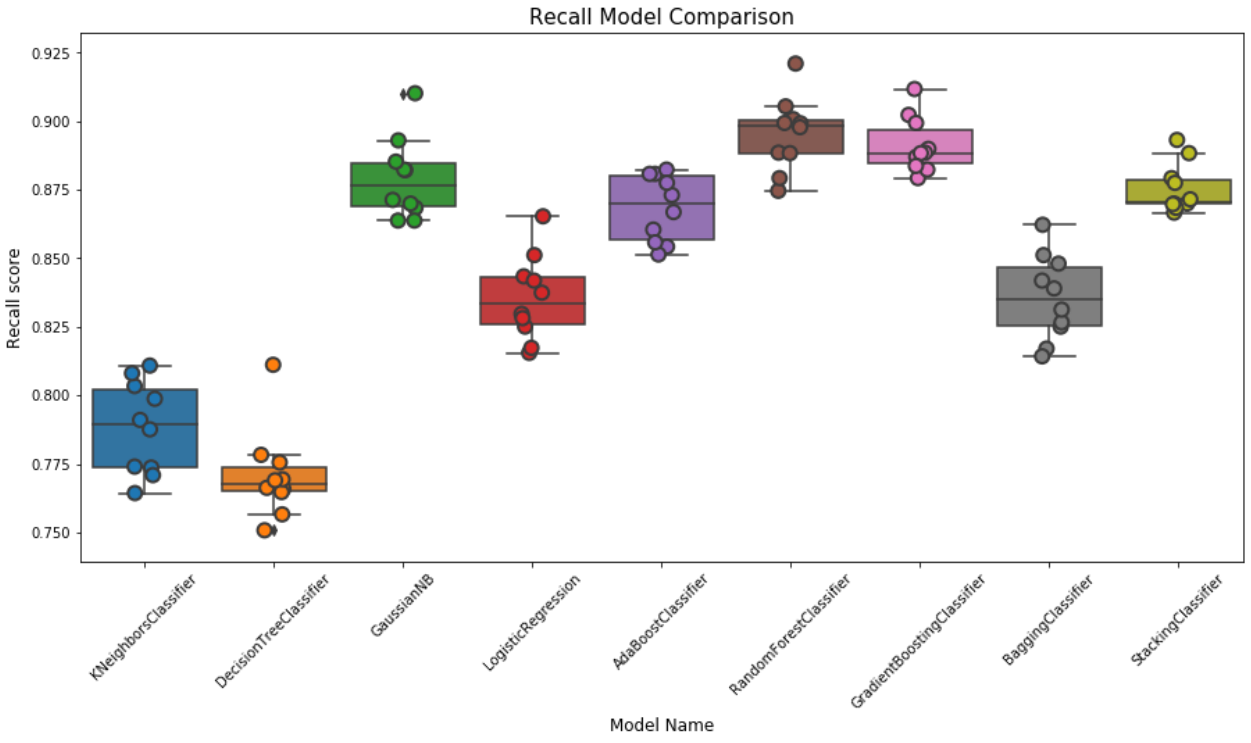

Score Time Model Comparison
Score Time score
0.10
0.08
0.06
0.04
0.02
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name

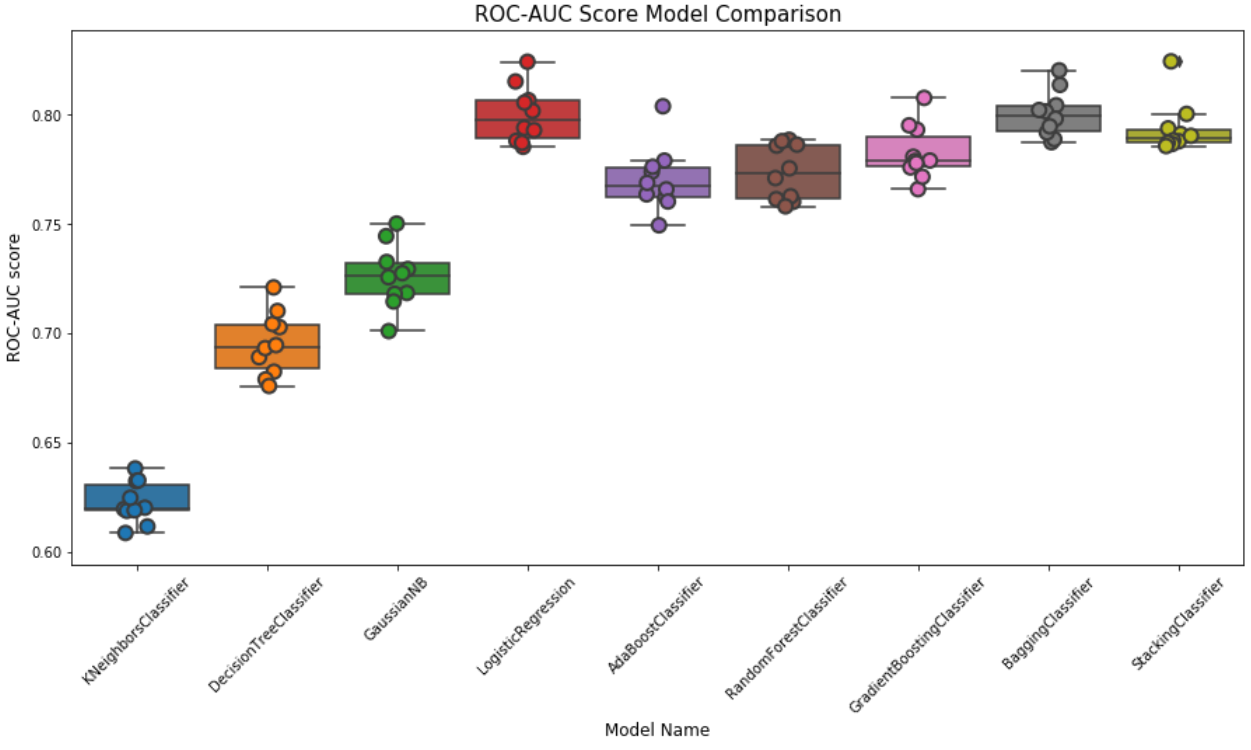

Balanced Accuracy Model Comparison
Balanced Accuracy score
0.80
0.75
0.70
0.65
0.60
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name


F1 Score Model Comparison
F1 score
0.88
0.86
0.84
0.82
0.80
0.78
0.76
0.74
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name


F2 Score Model Comparison
F2 score
0.90
0.88
0.86
0.84
0.82
0.80
0.78
0.76
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name


Precision Model Comparison
Precision score
0.875
0.850
0.825
0.800
0.775
0.750
0.725
0.700
KNeighborsClassifier
DecisionTreeClassifier
GaussianNB
LogisticRegression
AdaBoostClassifier
RandomForestClassifier
GradientBoostingClassifier
BaggingClassifier
StackingClassifier
Model Name





Receiver Operating Characteristic (ROC) Curves ,Precision-Recall Curves, Learning Curves, and Confusion Matrices

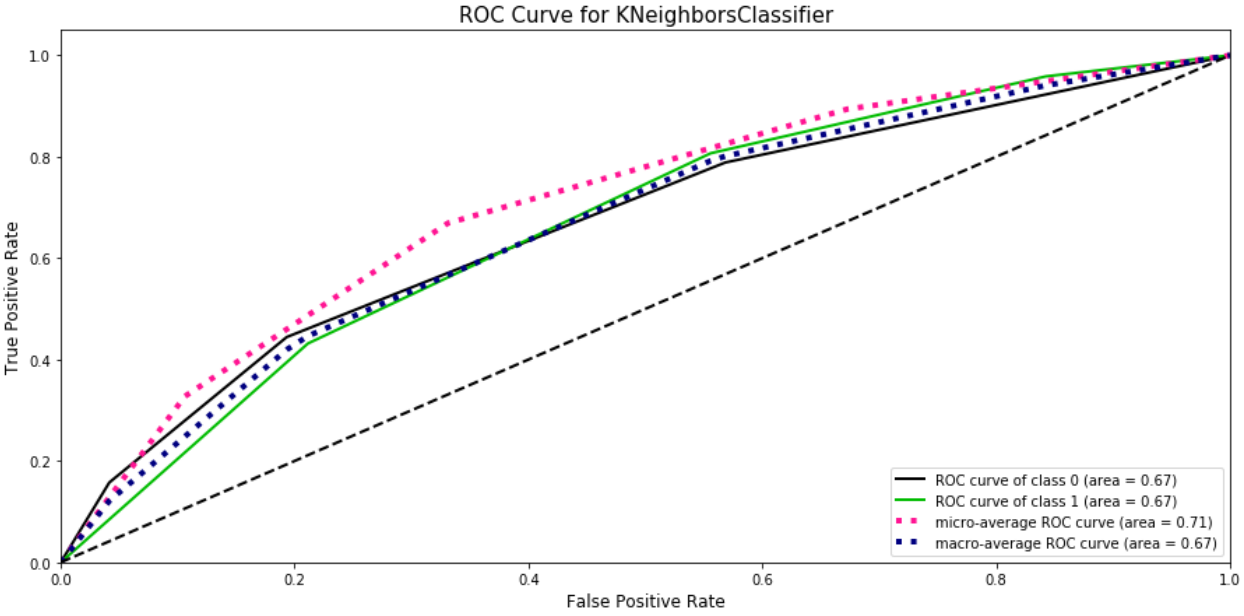
Below are generated ROC curves for the basic models, trained using training data (with 5-fold Cross Validation), and tested with testing data. The ROC curves for both training and testing data are generated onto the same plot for each model.

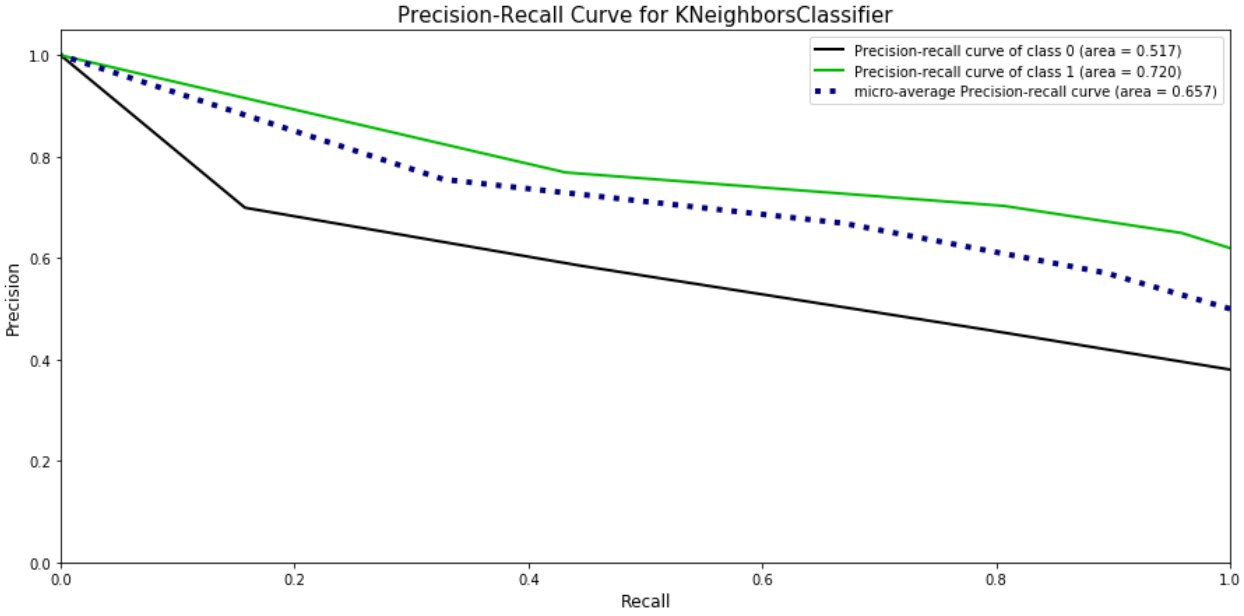
Also generated are Precision-Recall curves for the basic models as well. Similarly, the Precision-Recall curves for both training and testing data are generated onto the same plot for each model.

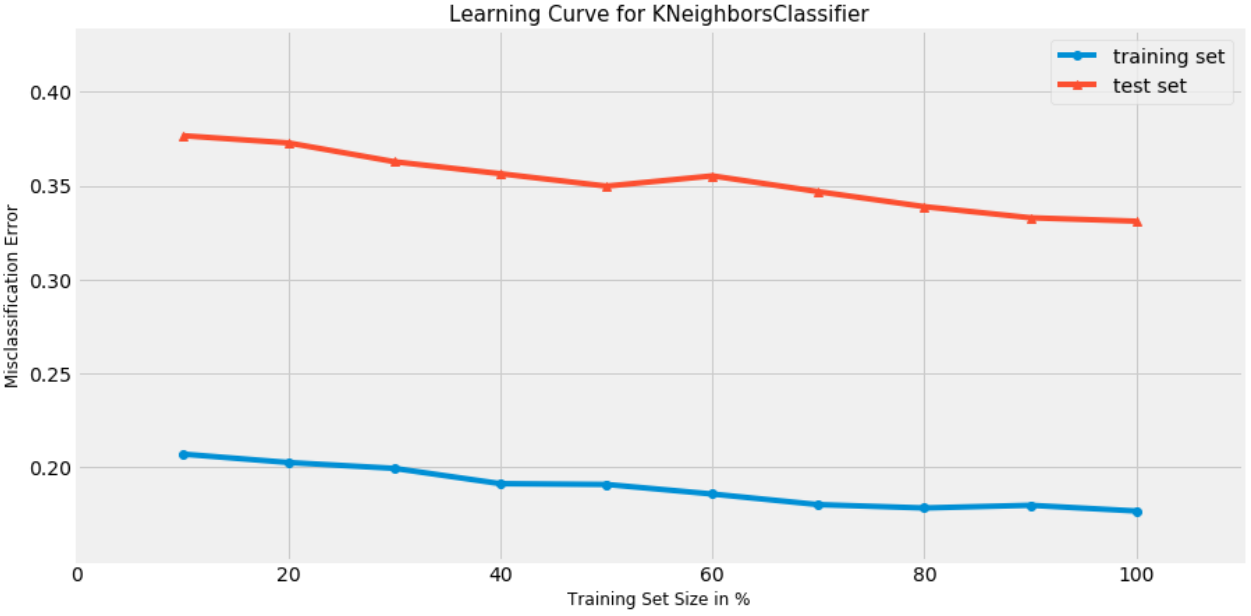
In [12]:
```
#predictions_entries = []

for model in models:
    model_name = model.__class__.__name__

    Y_train_predictions = cross_val_predict(
        model,
        X_train,
        Y_train_1d,
        cv = cv_num,
        method = 'predict_proba',
        n_jobs=-1
    )

    trained_model = model.fit(X_train, Y_train_1d)
    Y_test_predictions = trained_model.predict_proba(X_test)
    Y_test_binary_predictions = trained_model.predict(X_test)

    #ROC
    #train
    # Compute ROC curve and ROC area for each class
    fpr_train = dict()
    tpr_train = dict()
    roc_auc_train = dict()

    for i in range(classes_num):
        fpr_train[i], tpr_train[i], _ = roc_curve(Y_train_np_array[:, 0], Y_train_predictions[:, i])
        roc_auc_train[i] = auc(fpr_train[i], tpr_train[i])

    # Compute micro-average ROC curve and ROC area
    #we only need to provide probability estaimates of the positive class (i.e. Y_train_predictions[:,
1])
    fpr_train["micro"], tpr_train["micro"], _ = roc_curve(Y_train_np_array.ravel(), Y_train_prediction
s[:, 1].ravel())
    roc_auc_train["micro"] = auc(fpr_train["micro"], tpr_train["micro"])

    #test
    # Compute ROC curve and ROC area for each class
    fpr_test = dict()
    tpr_test = dict()
    roc_auc_test = dict()

    for i in range(classes_num):
        fpr_test[i], tpr_test[i], _ = roc_curve(Y_test_np_array[:, 0], Y_test_predictions[:, i])
        roc_auc_test[i] = auc(fpr_test[i], tpr_test[i])

    # Compute micro-average ROC curve and ROC area
    #we only need to provide probability estaimates of the positive class (i.e. Y_train_predictions[:,
1])
    fpr_test["micro"], tpr_test["micro"], _ = roc_curve(Y_test_np_array.ravel(), Y_test_predictions[:,
1].ravel())
    roc_auc_test["micro"] = auc(fpr_test["micro"], tpr_test["micro"])

    #plot roc curve
    #this manually plots roc curve train vs. test
    #line_weight_num = 2.5

    #plt.figure(figsize=fig_size_tuple)
    #plt.plot(fpr_test[1], tpr_test[1], color='red',
    #         lw = line_weight_num, label='ROC curve - Test (area = %0.3f)' % roc_auc_test[1])
    #plt.plot(fpr_train[1], tpr_train[1], color='darkorange',
    #         lw = line_weight_num, label='ROC curve - Train (area = %0.3f)' % roc_auc_train[1])
    #plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    #plt.xlim([0.0, 1.0])
    #plt.ylim([0.0, 1.05])
    #plt.xlabel('False Positive Rate', fontsize=label_fontsize_num)
    #plt.ylabel('True Positive Rate', fontsize=label_fontsize_num)
    #plt.title('Receiver operating characteristic for ' + model_name, fontsize=title_fontsize_num)
    #plt.legend(loc="lower right")
    #plt.show()
```

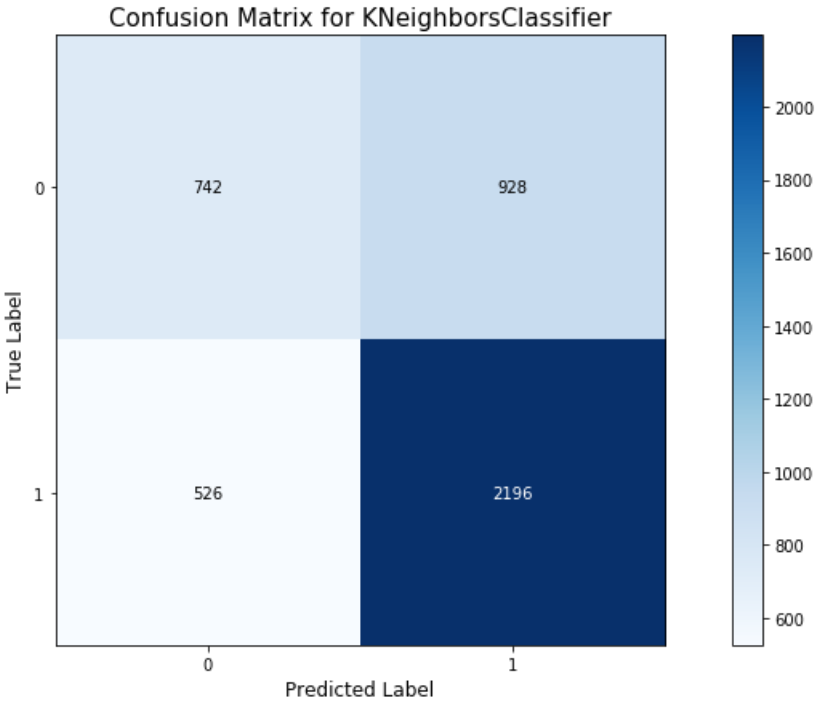
```
    #uses scikit-plot instead of plotting manually
    plot_roc(Y_test, Y_test_predictions,
            figsize=fig_size_tuple, title_fontsize=title_fontsize_num, text_fontsize=10, title='ROC Cu
rve for ' + model_name)
    plt.xlabel('False Positive Rate', fontsize=label_fontsize_num)
    plt.ylabel('True Positive Rate', fontsize=label_fontsize_num)
    plt.show()


    #Precision-Recall
    #average precision-recall score
    average_precision_train = average_precision_score(Y_train, Y_train_predictions[:, 1])
    average_precision_test = average_precision_score(Y_test, Y_test_predictions[:, 1])

    #rint('Average train precision-recall score: {0:0.3f}'.format(average_precision_train))
    #rint('Average test precision-recall score: {0:0.3f}'.format(average_precision_test))

    #calculate precision recall curve
    precision_train = {}
    precision_test = {}
    recall_train = {}
    recall_test = {}

    precision_train, recall_train, _ = precision_recall_curve(Y_train_np_array.ravel(), Y_train_predic
tions[:, 1].ravel())
    precision_test, recall_test, _ = precision_recall_curve(Y_test_np_array.ravel(), Y_test_prediction
s[:, 1].ravel())

    #plot preciision-recall curve
    #this manually plots precision-recall curve train vs. test
    #plt.figure(figsize=fig_size_tuple)
    #plt.step(recall_train, precision_train, where='post', color='darkblue',
    #         label='Precision-Recall Curve - Train (score = %0.3f)' % average_precision_train)
    #plt.step(recall_test, precision_test, where='post', color='lightblue',
    #         label='Precision-Recall Curve - Test (score = %0.3f)' % average_precision_test)
    #plt.xlabel('Recall', fontsize=label_fontsize_num)
    #plt.ylabel('Precision', fontsize=label_fontsize_num)
    #plt.ylim([0.0, 1.05])
    #plt.xlim([0.0, 1.0])
    #plt.title(
    #    'Precision-Recall Curve for' + model_name, fontsize=title_fontsize_num)
    #plt.legend(loc="lower right")
    #plt.show()

    #this plots a shitty precision recall curve that can't really be modified visually
    #so don't use it unless if you have to
    #disp = plot_precision_recall_curve(model, X_test, Y_test)
    #disp.ax_.set_title('2-class Precision-Recall curve: AP={0:0.2f}'.format(average_precision_test))

    #uses scikit-plot instead of plotting manually
    plot_precision_recall(Y_test, Y_test_predictions,
                          figsize=fig_size_tuple, title_fontsize=title_fontsize_num, text_fontsize=10,
title='Precision-Recall Curve for ' + model_name)
    plt.xlabel('Recall', fontsize=label_fontsize_num)
    plt.ylabel('Precision', fontsize=label_fontsize_num)
    plt.show()

    #plot learning curve
    #this one uses scikitplot to plot a usable learning curve
    #unfortunately we can't really choose a misclassification error for the scoring, so we use F2 inst
ead...not ideal but oh well
    #plot_learning_curve(model, X_test, Y_test_1d, cv = cv_num, random_state = random_state_num,
    #                    scoring = f2_scorer,
    #                    figsize=fig_size_tuple, title_fontsize=title_fontsize_num, text_fontsize=10,
 title='Learning Curve for ' + model_name)
    #plt.show()

    #this one uses mlxtend to plot a better learning curve
    #this is better because it offers misclassification error for the scoring, so more useful
    plt.figure(figsize=fig_size_tuple)
    plot_learning_curves(X_train, Y_train_1d, X_test, Y_test_1d, model)
    plt.title('Learning Curve for ' + model_name, fontsize=title_fontsize_num)
```

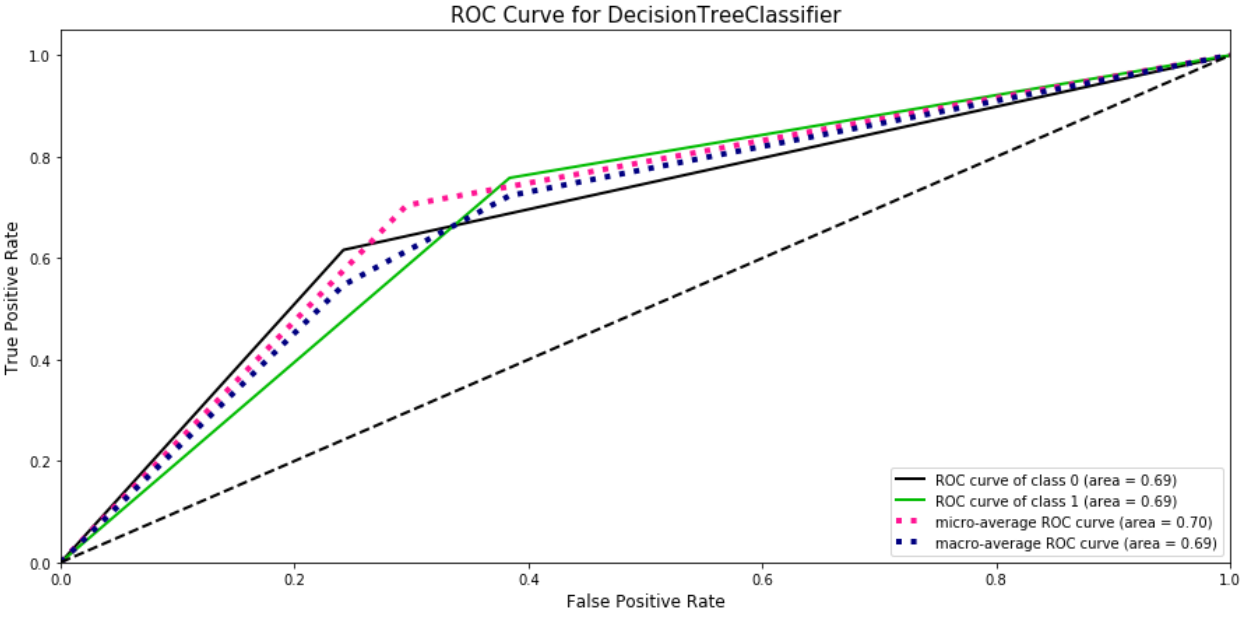
```
    plt.xlabel('Training Set Size in %', fontsize=label_fontsize_num)
    plt.ylabel('Misclassification Error', fontsize=label_fontsize_num)
    plt.show()


    #confusion matrices
    plot_confusion_matrix(Y_test, Y_test_binary_predictions,
                          figsize=fig_size_tuple, title_fontsize=title_fontsize_num, text_fontsize=10,
title='Confusion Matrix for ' + model_name)
    plt.xlabel('Predicted Label', fontsize=label_fontsize_num)
    plt.ylabel('True Label', fontsize=label_fontsize_num)
    plt.show()


    #decision regions
    #not needed at this time, we need to reduce dimensions somehow in order to use decision trees in a
useful manner
```

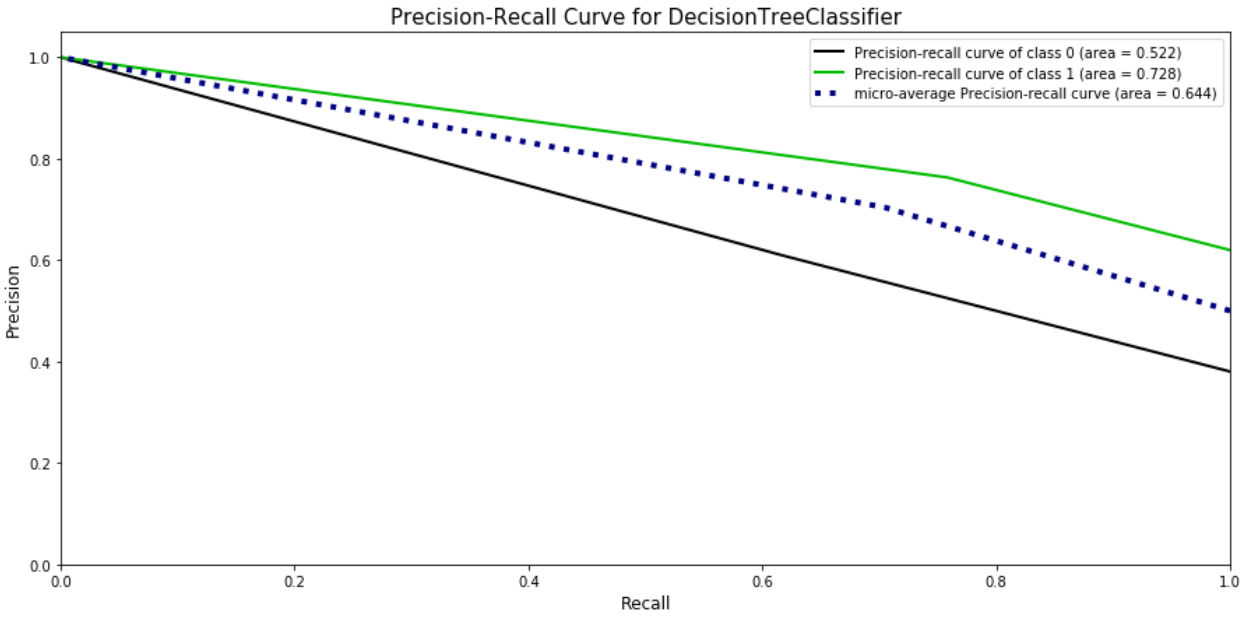

ROC Curve for KNeighborsClassifier
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 0.67)
ROC curve of class 1 (area = 0.67)
micro-average ROC curve (area = 0.71)
macro-average ROC curve (area = 0.67)

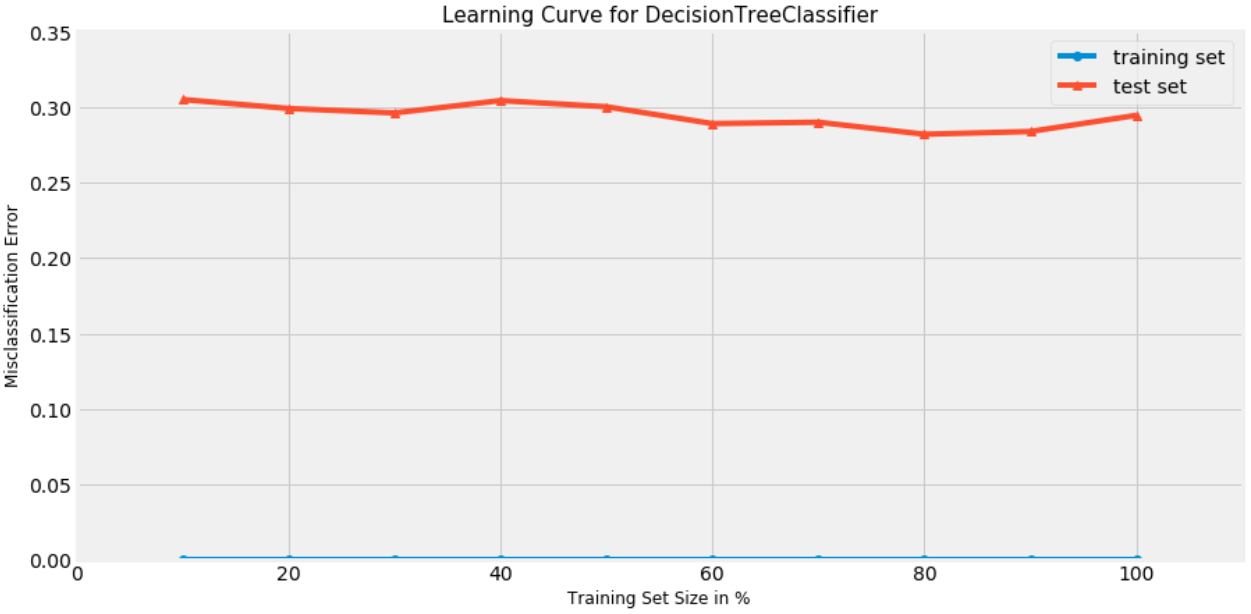

Precision-Recall Curve for KNeighborsClassifier
Precision
Recall
Precision-recall curve of class 0 (area = 0.517)
Precision-recall curve of class 1 (area = 0.720)
micro-average Precision-recall curve (area = 0.657)

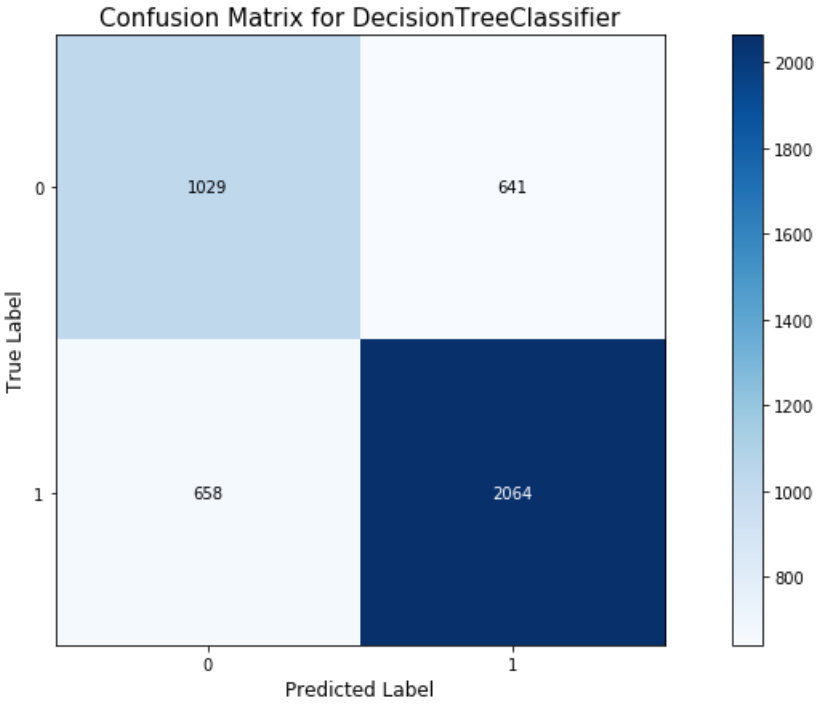

Learning Curve for KNeighborsClassifier
Misclassification Error
Training Set Size in %
training set
test set

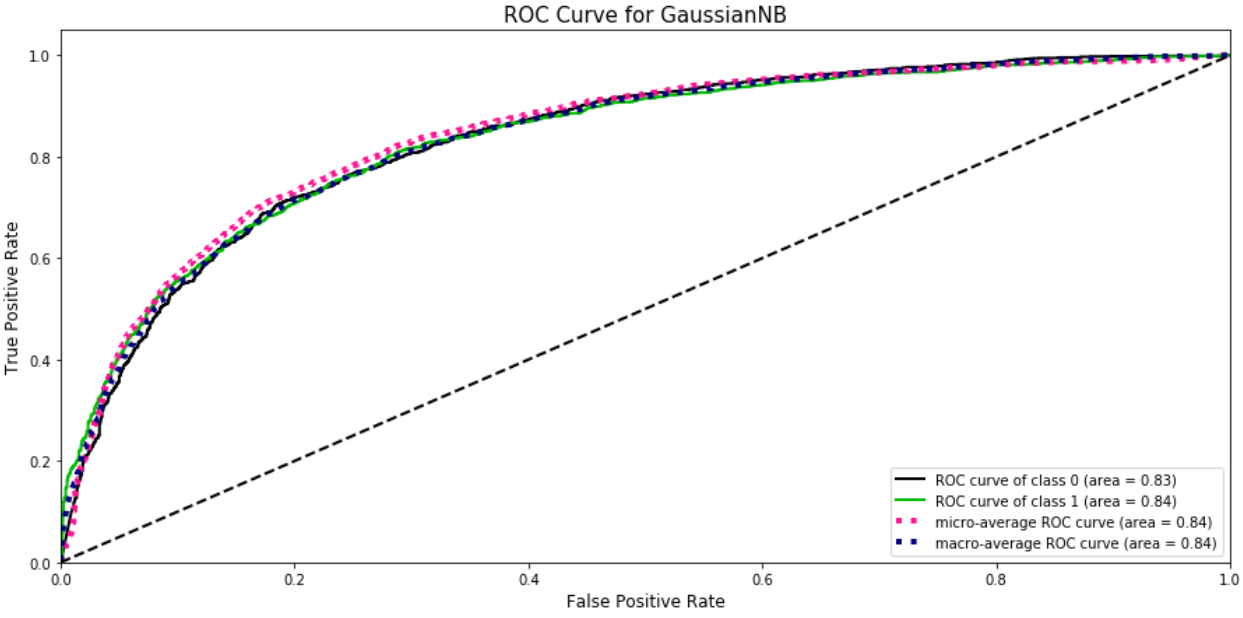

Confusion Matrix for KNeighborsClassifier
742
928
526
2196
0
1
True Label
0
1
Predicted Label
2000
1800
1600
1400
1200
1000
800
600

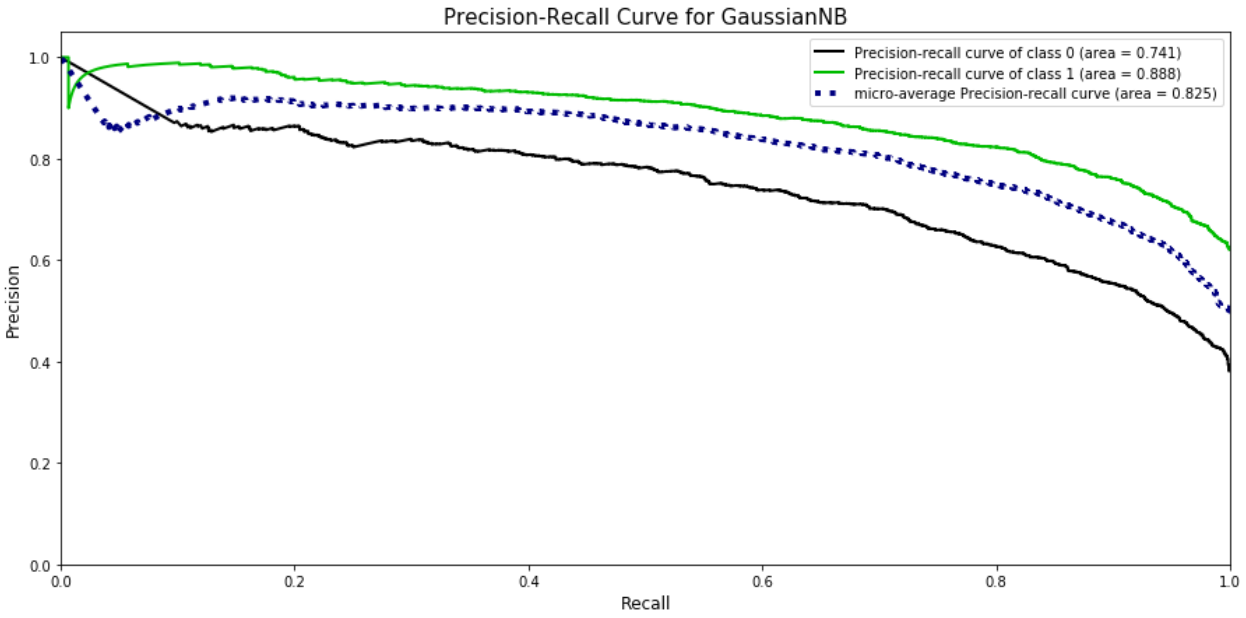

ROC Curve for DecisionTreeClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.69)
ROC curve of class 1 (area = 0.69)
micro-average ROC curve (area = 0.70)
macro-average ROC curve (area = 0.69)

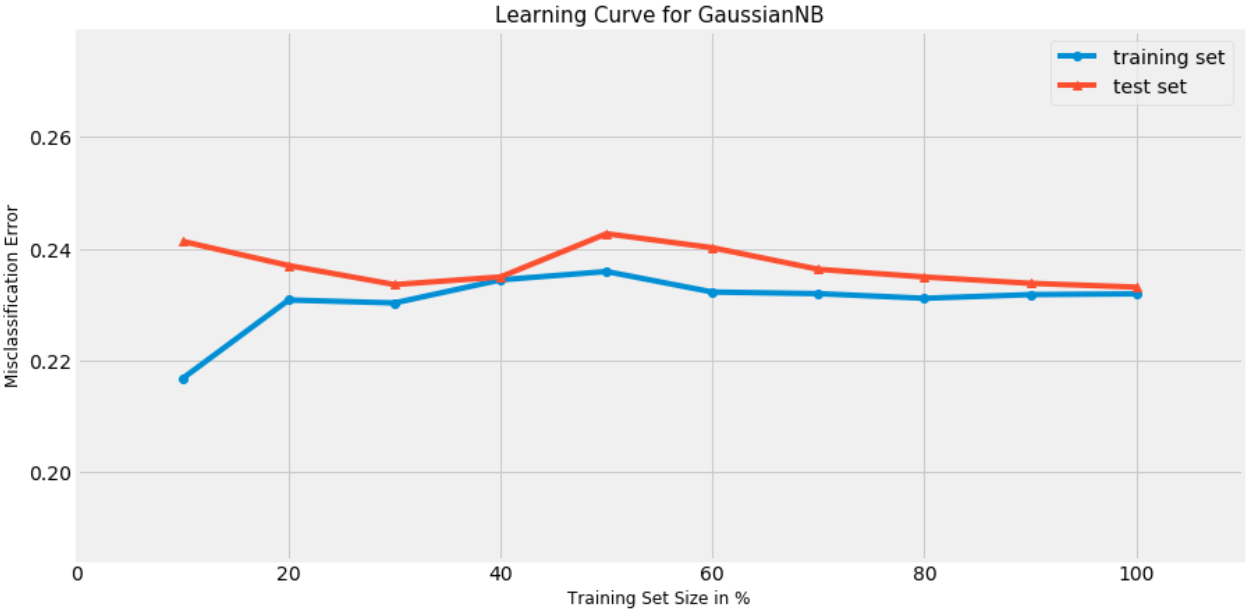

Precision-Recall Curve for DecisionTreeClassifier
Precision-recall curve of class 0 (area = 0.522)
Precision-recall curve of class 1 (area = 0.728)
micro-average Precision-recall curve (area = 0.644)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

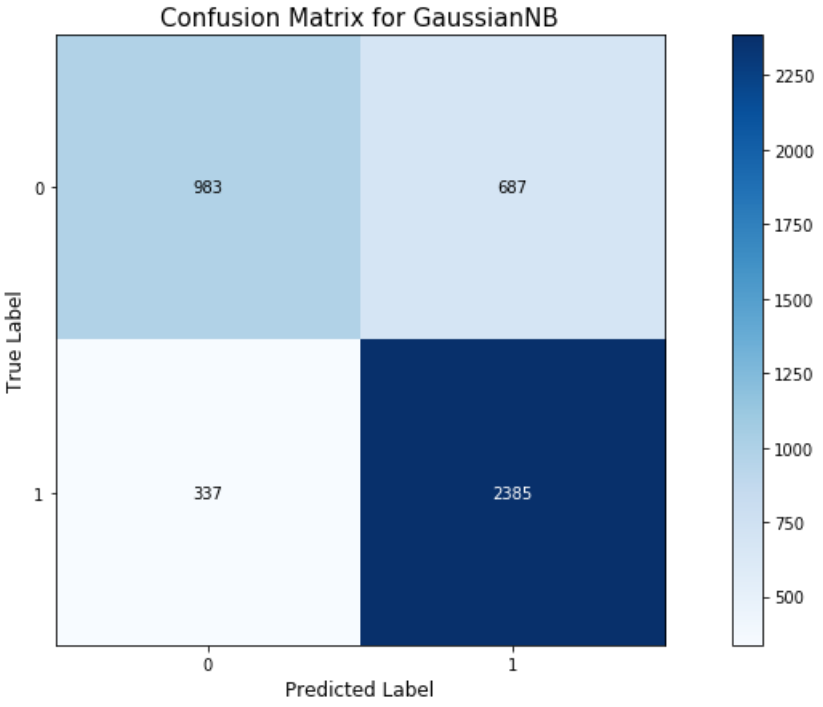

Learning Curve for DecisionTreeClassifier
training set
test set
Misclassification Error
Training Set Size in %
0.00
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0
20
40
60
80
100

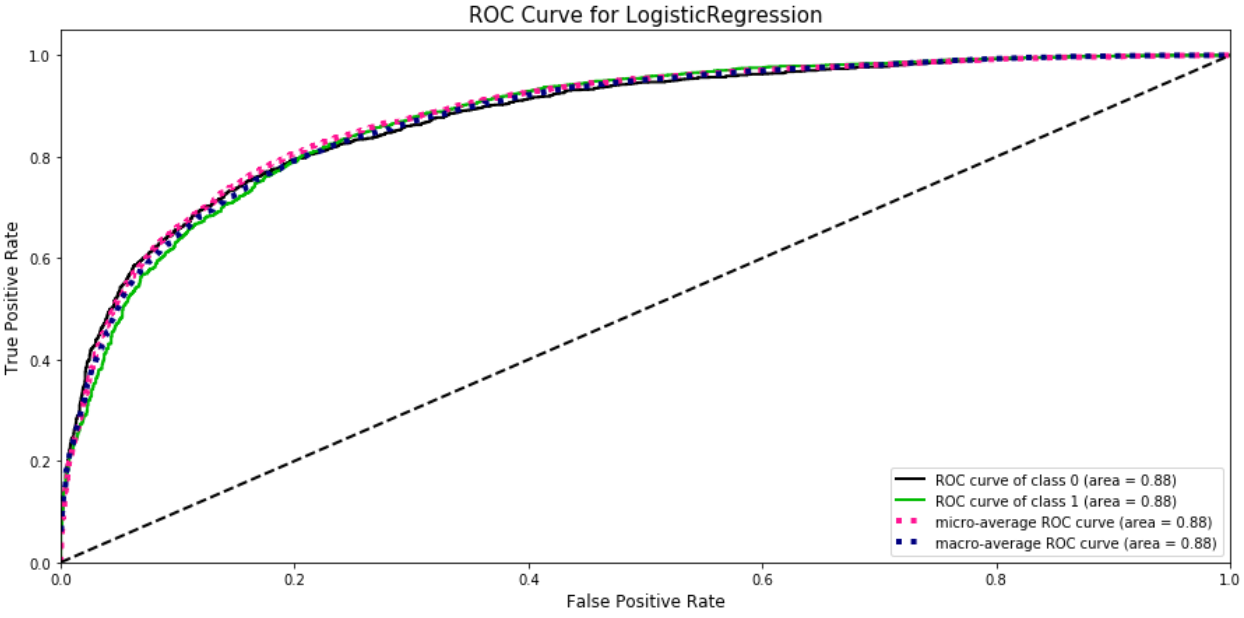

Confusion Matrix for DecisionTreeClassifier
1029
641
658
2064
0
1
True Label
0
1
Predicted Label
2000
1800
1600
1400
1200
1000
800

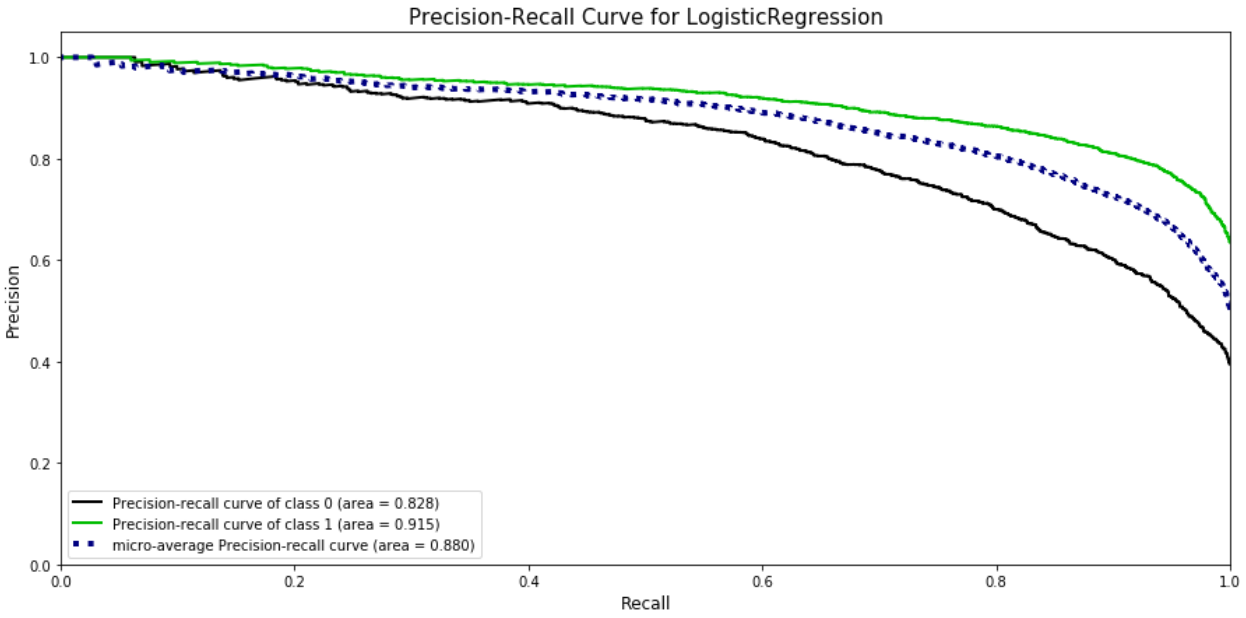

ROC Curve for GaussianNB
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.83)
ROC curve of class 1 (area = 0.84)
micro-average ROC curve (area = 0.84)
macro-average ROC curve (area = 0.84)

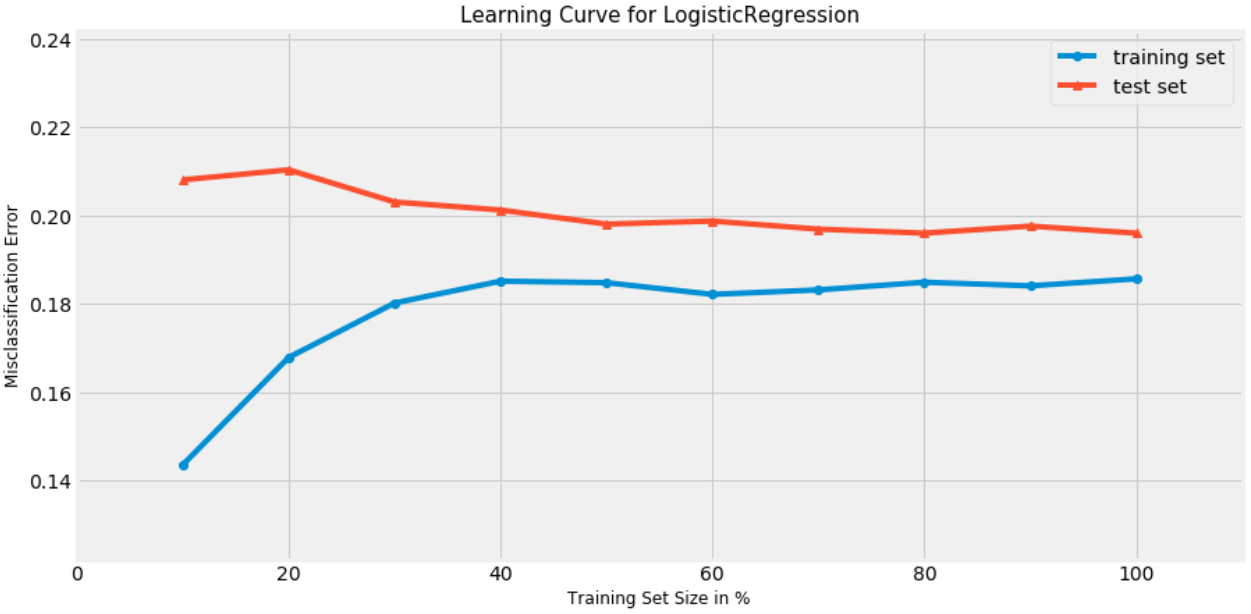

Precision-Recall Curve for GaussianNB
Precision-recall curve of class 0 (area = 0.741)
Precision-recall curve of class 1 (area = 0.888)
micro-average Precision-recall curve (area = 0.825)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

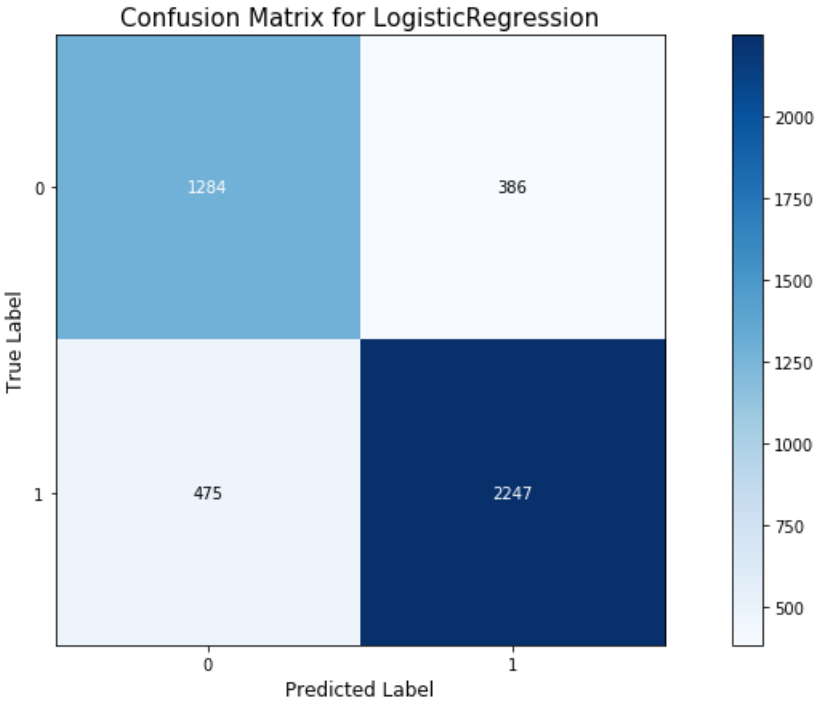

Learning Curve for GaussianNB
training set
test set
Misclassification Error
Training Set Size in %
0.20
0.22
0.24
0.26
0
20
40
60
80
100

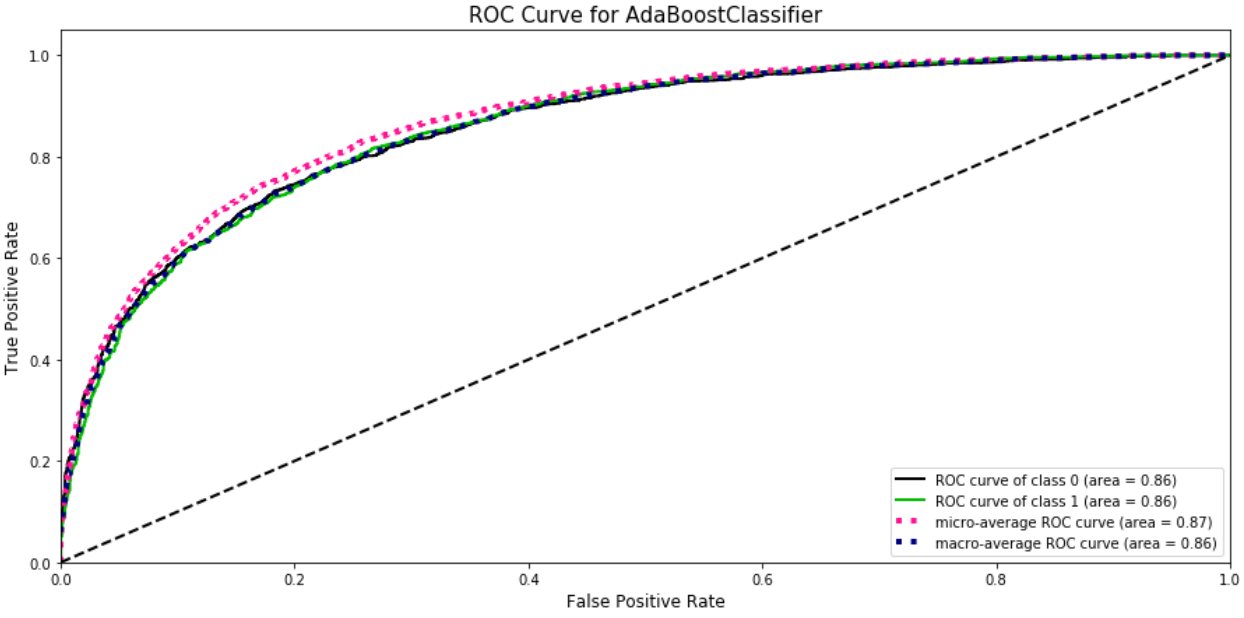

Confusion Matrix for GaussianNB
983
687
337
2385
0
1
True Label
0
1
Predicted Label
2250
2000
1750
1500
1250
1000
750
500

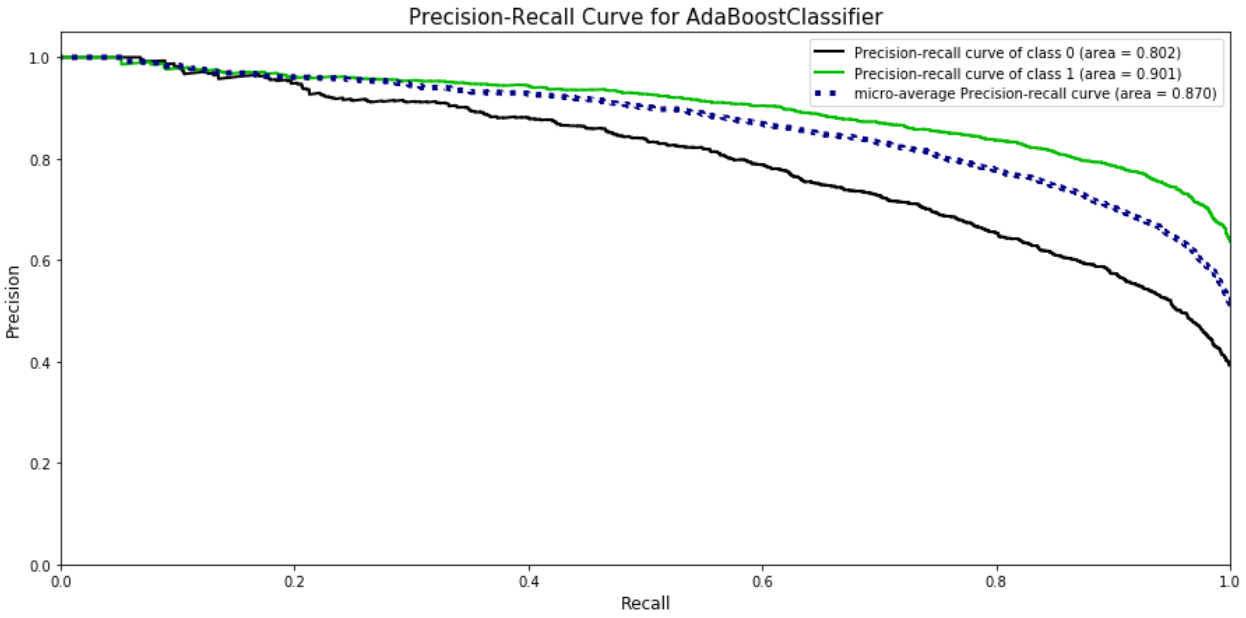

ROC Curve for LogisticRegression
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.88)
ROC curve of class 1 (area = 0.88)
micro-average ROC curve (area = 0.88)
macro-average ROC curve (area = 0.88)

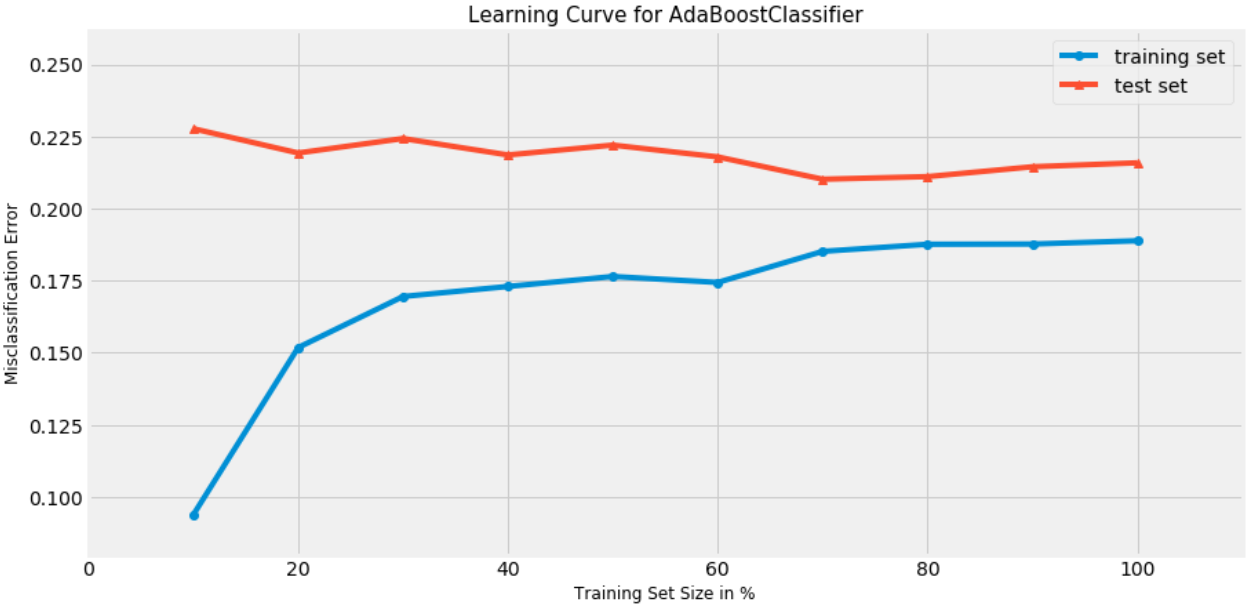

Precision-Recall Curve for LogisticRegression
Precision
Recall
Precision-recall curve of class 0 (area = 0.828)
Precision-recall curve of class 1 (area = 0.915)
micro-average Precision-recall curve (area = 0.880)

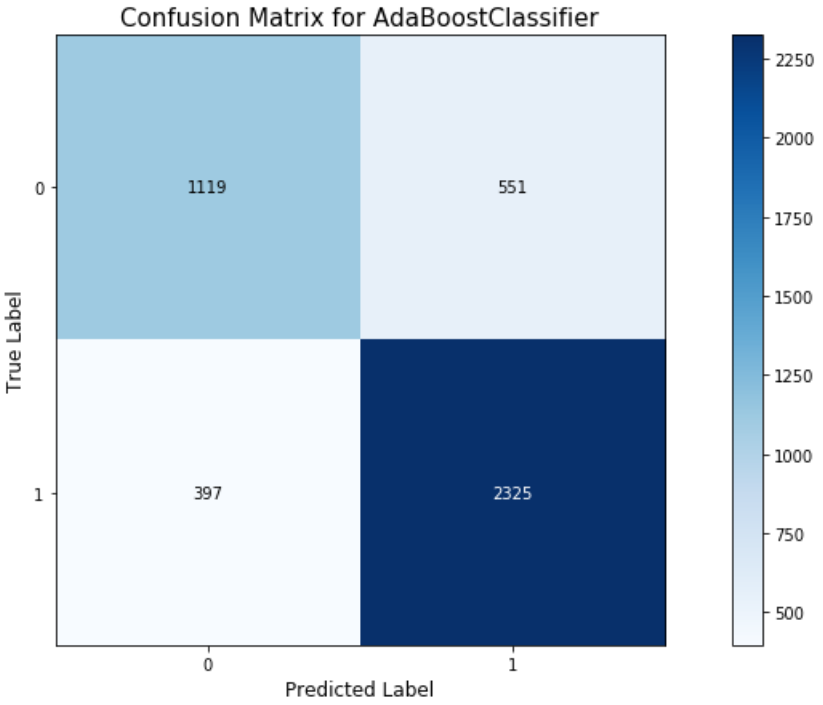

Learning Curve for LogisticRegression
Misclassification Error
Training Set Size in %
training set
test set

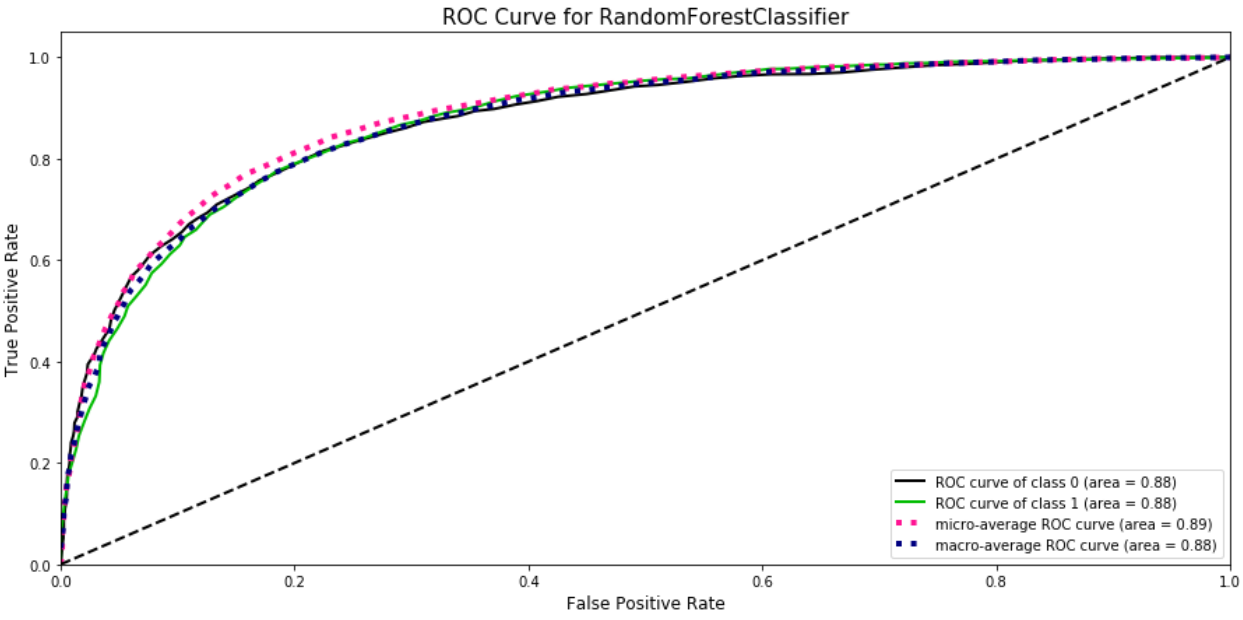

Confusion Matrix for LogisticRegression
1284
386
475
2247
0
1
True Label
0
1
Predicted Label
2000
1750
1500
1250
1000
750
500

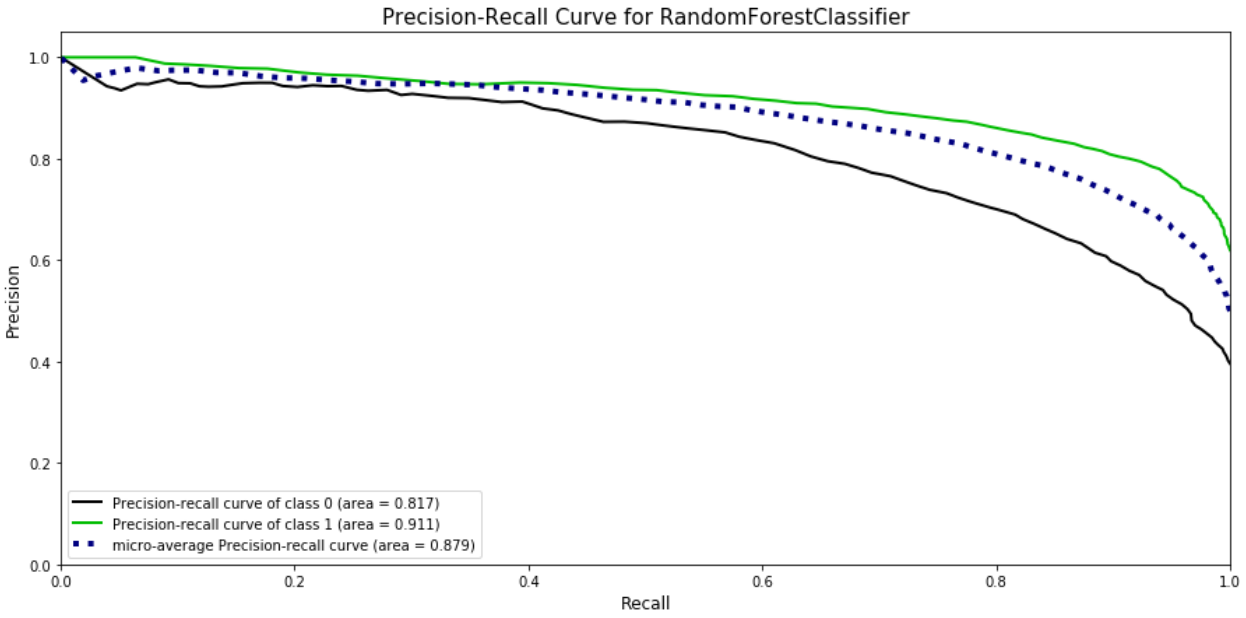

ROC Curve for AdaBoostClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.86)
ROC curve of class 1 (area = 0.86)
micro-average ROC curve (area = 0.87)
macro-average ROC curve (area = 0.86)

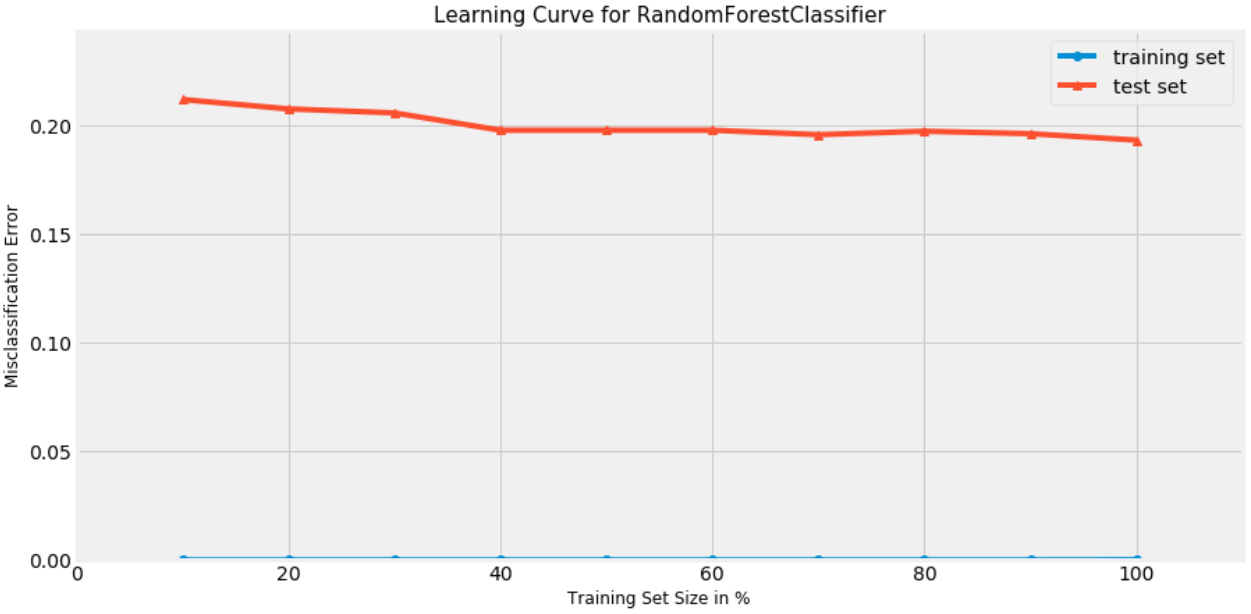

Precision-Recall Curve for AdaBoostClassifier
Precision-recall curve of class 0 (area = 0.802)
Precision-recall curve of class 1 (area = 0.901)
micro-average Precision-recall curve (area = 0.870)
Precision
Recall
0.0
0.2
0.4
0.6
0.8
1.0

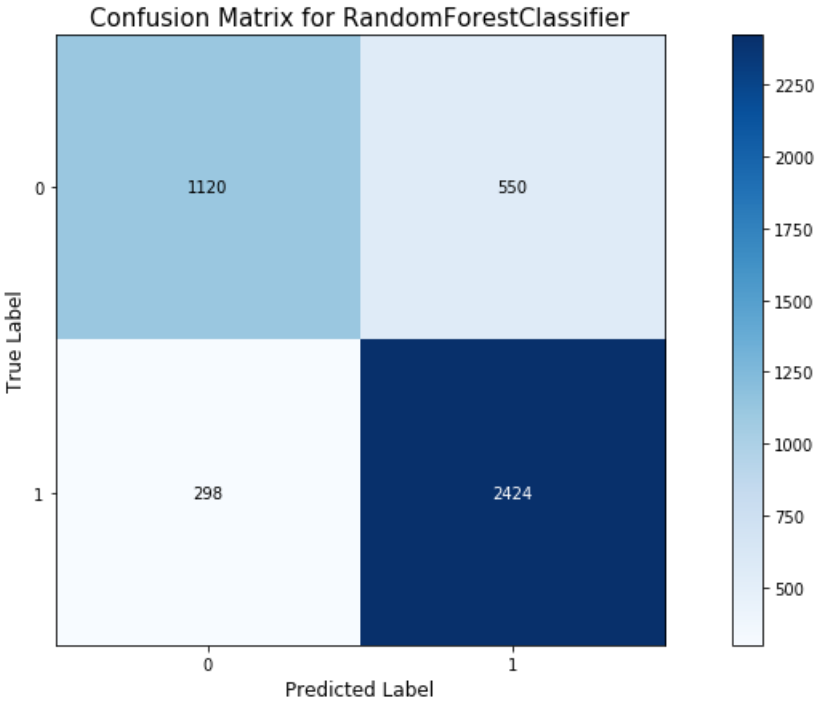

Learning Curve for AdaBoostClassifier
training set
test set
Misclassification Error
Training Set Size in %
0.100
0.125
0.150
0.175
0.200
0.225
0.250
0
20
40
60
80
100

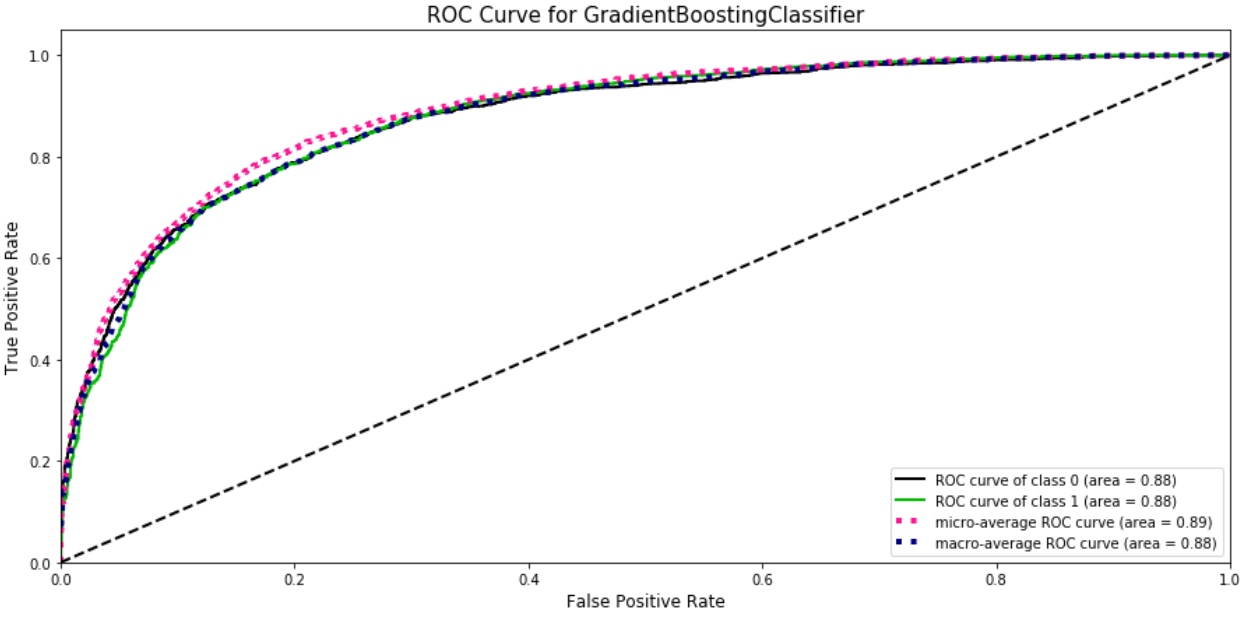

Confusion Matrix for AdaBoostClassifier
1119
551
397
2325
0
1
True Label
0
1
Predicted Label
2250
2000
1750
1500
1250
1000
750
500

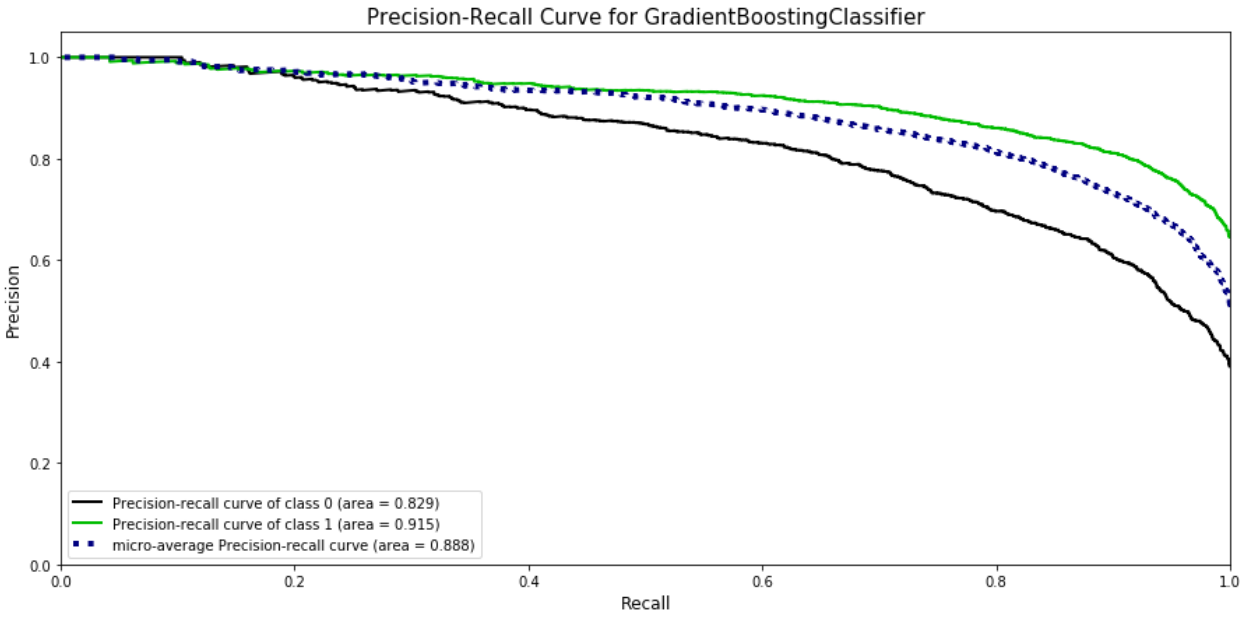

ROC Curve for RandomForestClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.88)
ROC curve of class 1 (area = 0.88)
micro-average ROC curve (area = 0.89)
macro-average ROC curve (area = 0.88)

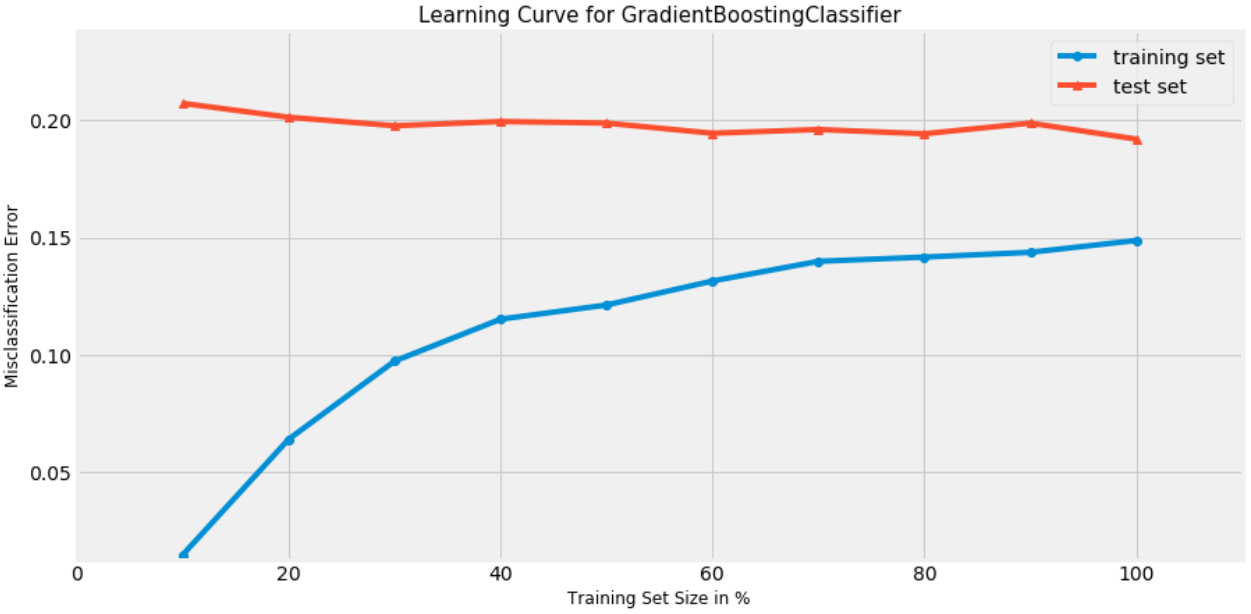

Precision-Recall Curve for RandomForestClassifier
Precision
Recall
Precision-recall curve of class 0 (area = 0.817)
Precision-recall curve of class 1 (area = 0.911)
micro-average Precision-recall curve (area = 0.879)

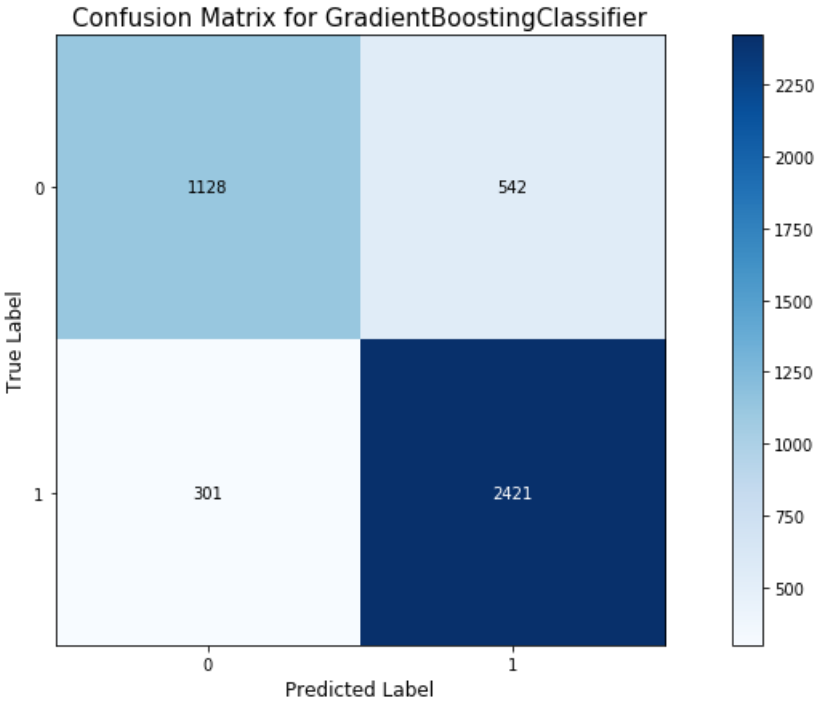

Learning Curve for RandomForestClassifier
training set
test set
Misclassification Error
Training Set Size in %

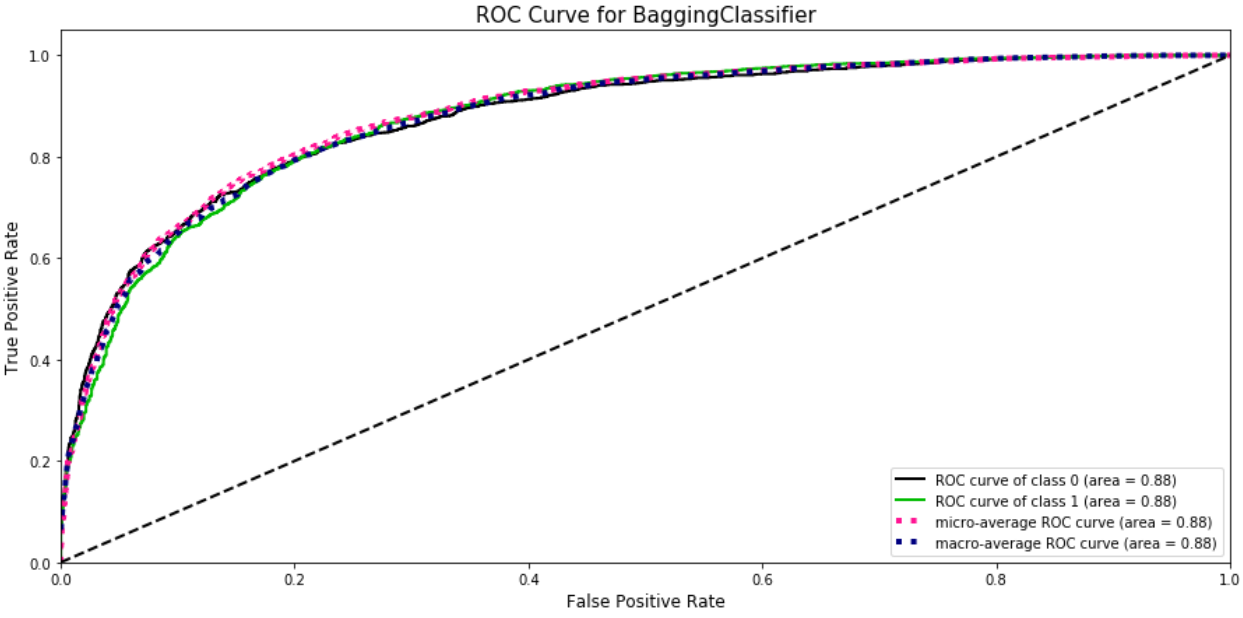

Confusion Matrix for RandomForestClassifier
1120
550
298
2424
0
1
True Label
0
1
Predicted Label
2250
2000
1750
1500
1250
1000
750
500

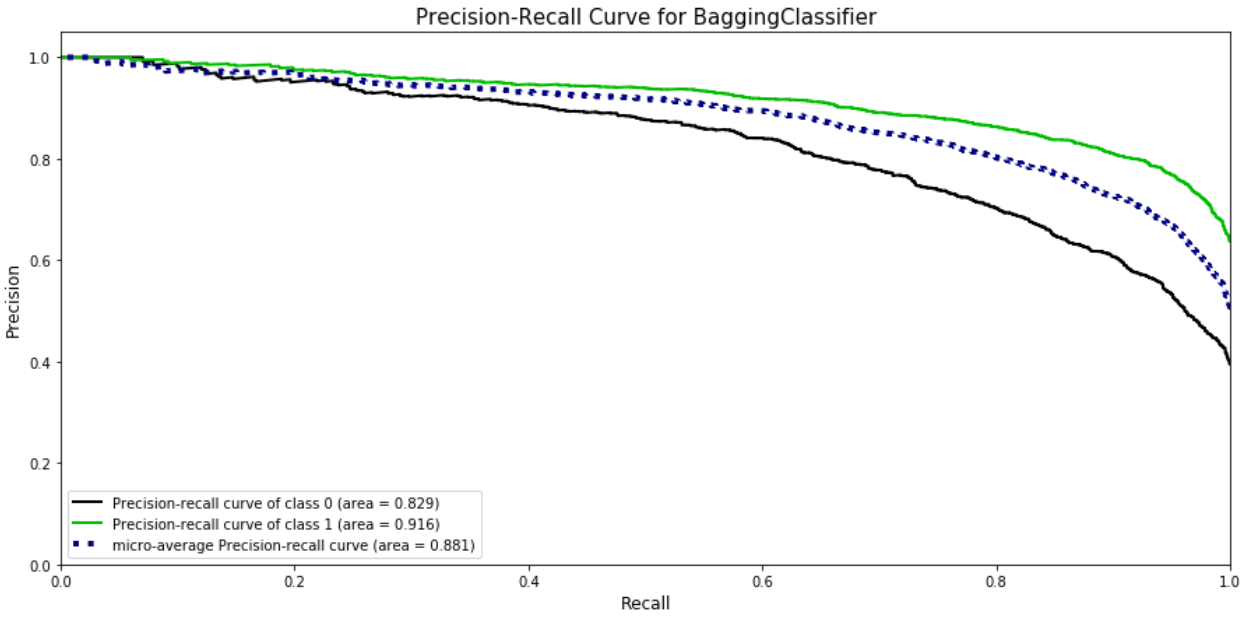

ROC Curve for GradientBoostingClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.88)
ROC curve of class 1 (area = 0.88)
micro-average ROC curve (area = 0.89)
macro-average ROC curve (area = 0.88)

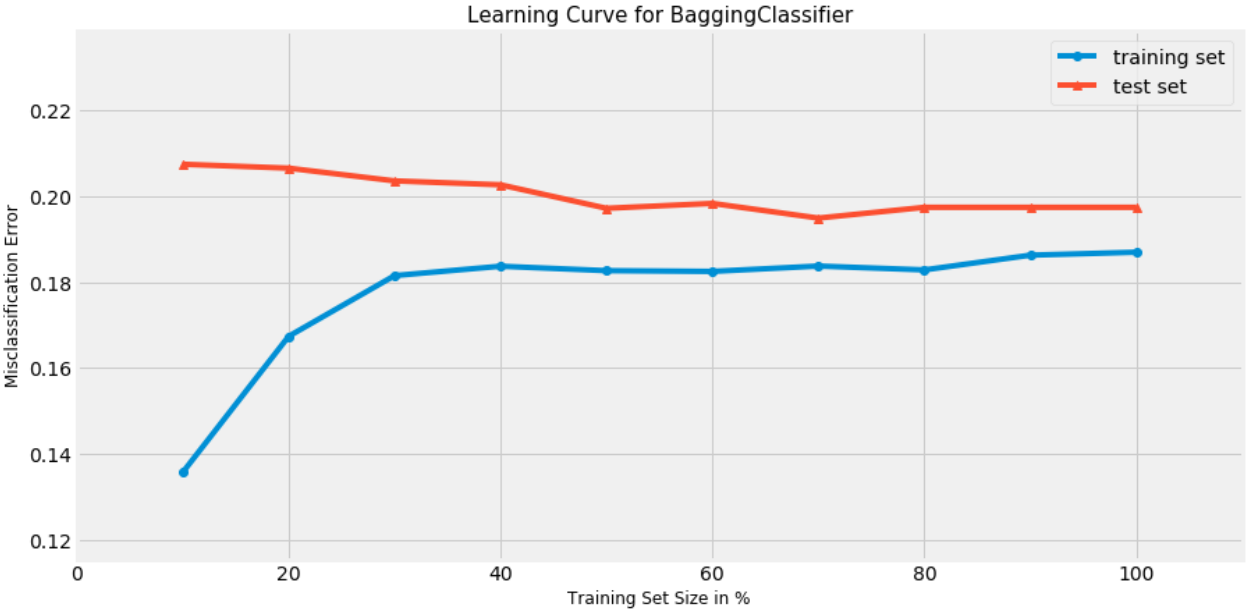

Precision-Recall Curve for GradientBoostingClassifier
Precision
Recall
Precision-recall curve of class 0 (area = 0.829)
Precision-recall curve of class 1 (area = 0.915)
micro-average Precision-recall curve (area = 0.888)

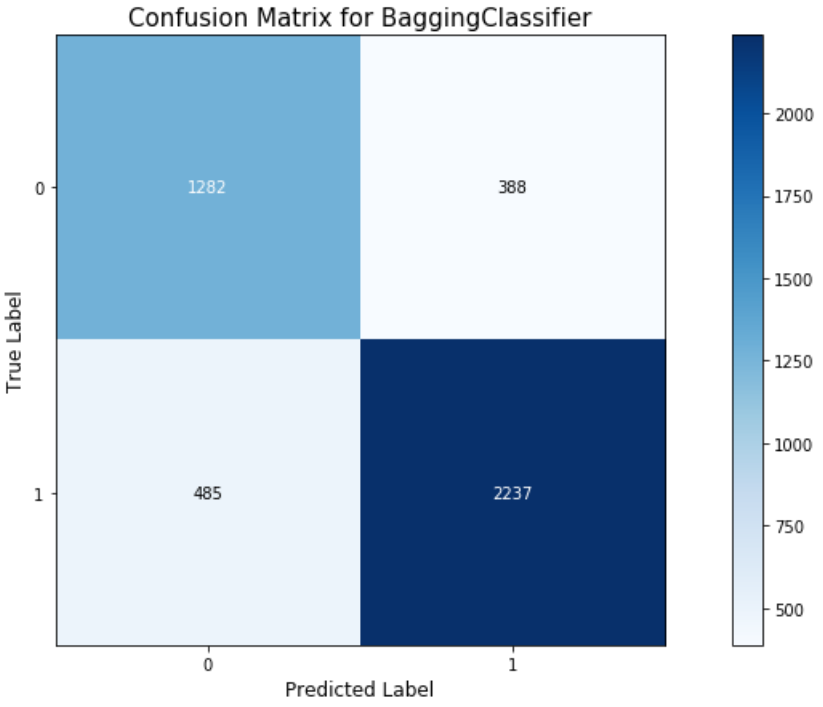

Learning Curve for GradientBoostingClassifier
Misclassification Error
Training Set Size in %
training set
test set

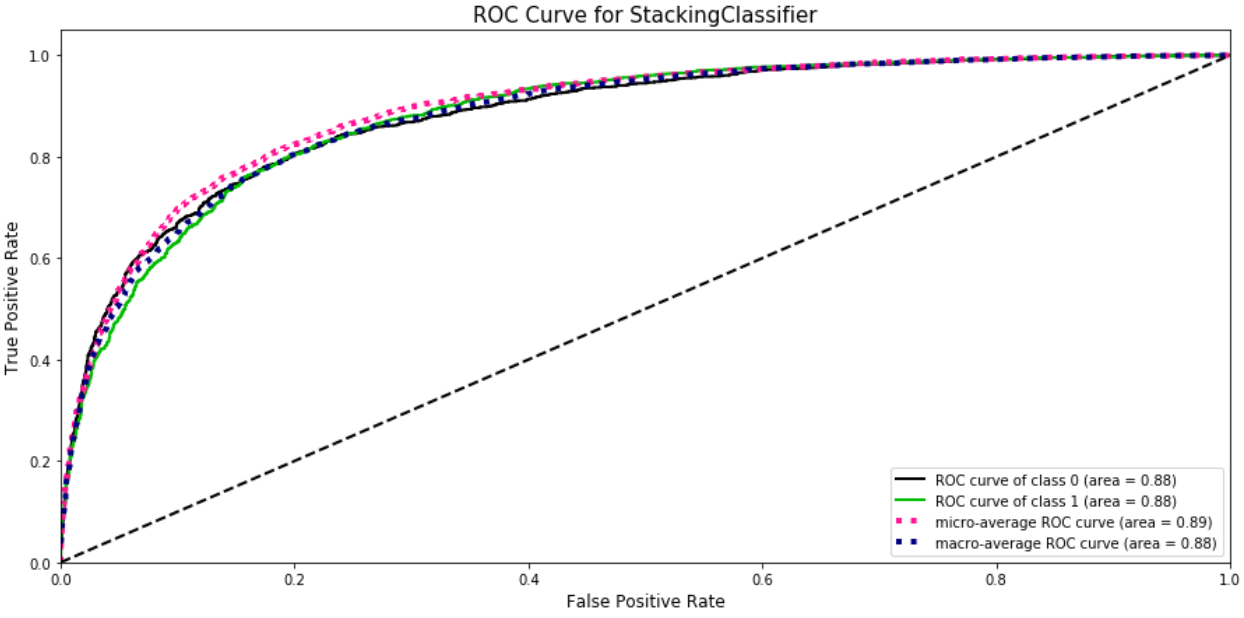

Confusion Matrix for GradientBoostingClassifier
1128
542
301
2421
0
1
True Label
0
1
Predicted Label
2250
2000
1750
1500
1250
1000
750
500

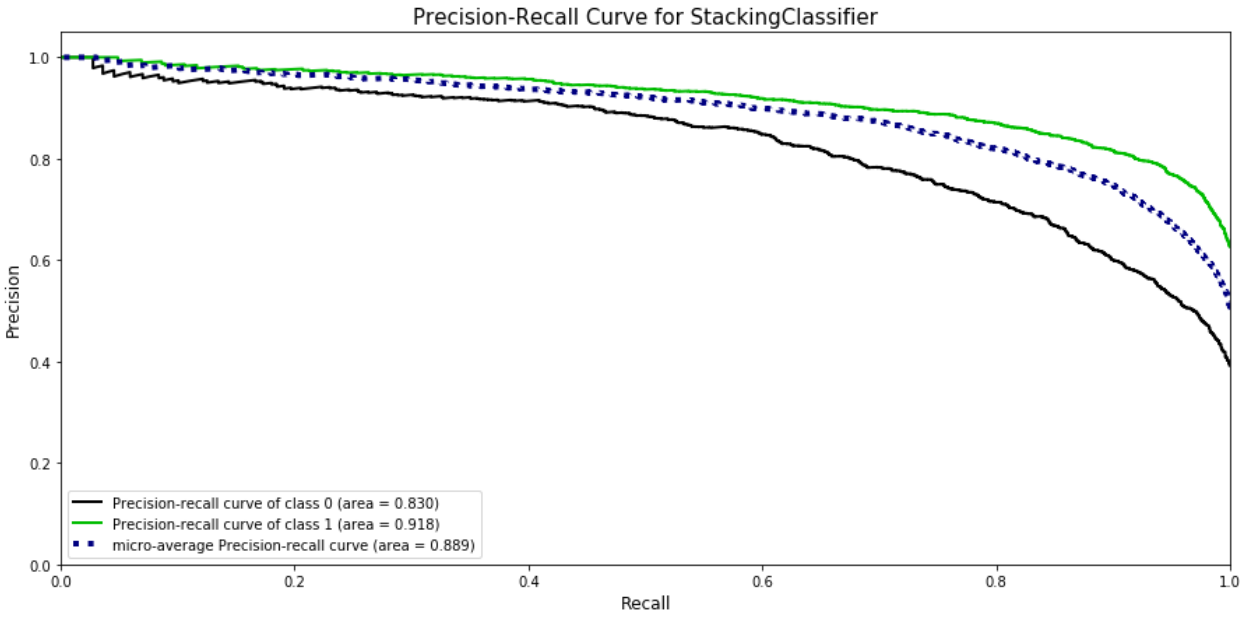

ROC Curve for BaggingClassifier
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve of class 0 (area = 0.88)
ROC curve of class 1 (area = 0.88)
micro-average ROC curve (area = 0.88)
macro-average ROC curve (area = 0.88)

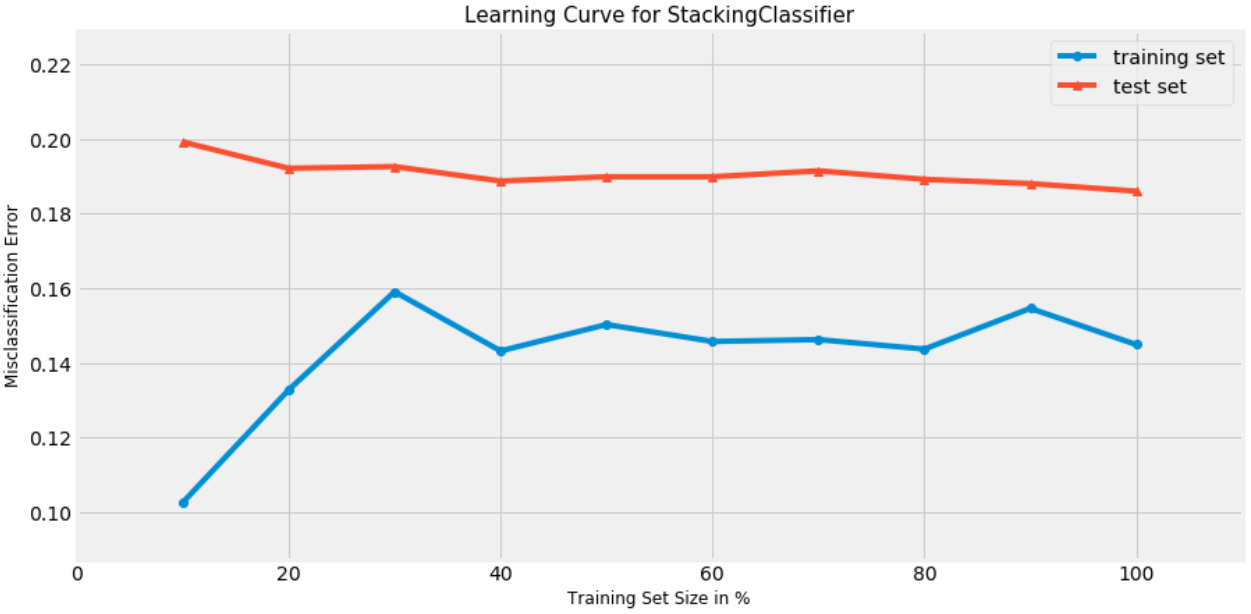

Precision-Recall Curve for BaggingClassifier
Precision
Recall
Precision-recall curve of class 0 (area = 0.829)
Precision-recall curve of class 1 (area = 0.916)
micro-average Precision-recall curve (area = 0.881)

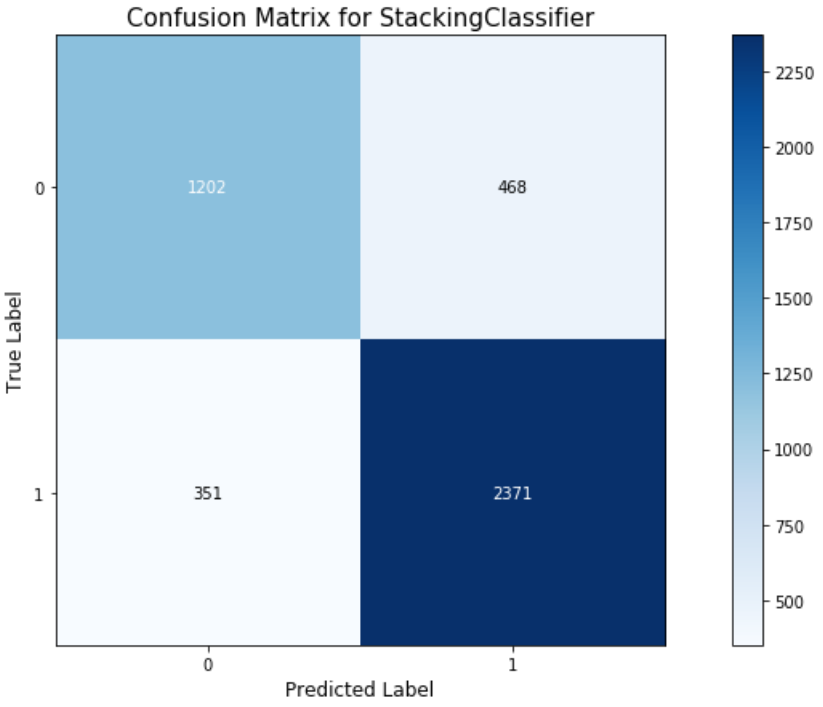

Learning Curve for BaggingClassifier
training set
test set
Misclassification Error
Training Set Size in %


Confusion Matrix for BaggingClassifier
1282
388
485
2237
True Label
Predicted Label
0
1
2000
1750
1500
1250
1000
750
500


ROC Curve for StackingClassifier
True Positive Rate
False Positive Rate
ROC curve of class 0 (area = 0.88)
ROC curve of class 1 (area = 0.88)
micro-average ROC curve (area = 0.89)
macro-average ROC curve (area = 0.88)


Precision-Recall Curve for StackingClassifier
Precision
Recall
Precision-recall curve of class 0 (area = 0.830)
Precision-recall curve of class 1 (area = 0.918)
micro-average Precision-recall curve (area = 0.889)


Learning Curve for StackingClassifier
Misclassification Error
Training Set Size in %
training set
test set


Confusion Matrix for StackingClassifier
1202
468
351
2371
0
1
True Label
0
1
Predicted Label
2250
2000
1750
1500
1250
1000
750
500

References

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Stacking Ensemble Method: https://towardsdatascience.com/stacking-made-easy-with-sklearn-e27a0793c92b (https://towardsdatascience.com/stacking-made-easy-with-sklearn-e27a0793c92b)
- Model Benchmarking: https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f (https://towardsdatascience.com/multi-class-text-classification-with-scikit-learn-12f1e60e0a9f)
- Model Benchmarking (Example Code): https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Consumer_complaints.ipynb (https://github.com/susanli2016/Machine-Learning-with-Python/blob/master/Consumer_complaints.ipynb)
- SciKit-Learn API Reference: https://scikit-learn.org/stable/modules/classes.html (https://scikit-learn.org/stable/modules/classes.html)
- Ensemble Methods: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205)
- Respective API Reference for each package used
- Hyperparameter Tuning:
  - https://scikit-learn.org/stable/modules/grid_search.html (https://scikit-learn.org/stable/modules/grid_search.html)
  - https://www.kaggle.com/tboyle10/hyperparameter-tuning (https://www.kaggle.com/tboyle10/hyperparameter-tuning)
  - https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5 (https://towardsdatascience.com/logistic-regression-model-tuning-with-scikit-learn-part-1-425142e01af5)
  - https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624 (https://towardsdatascience.com/hyperparameter-tuning-c5619e7e6624)
  - https://medium.com/@mandava807/cross-validation-and-hyperparameter-tuning-in-python-65cfb80ee485 (https://medium.com/@mandava807/cross-validation-and-hyperparameter-tuning-in-python-65cfb80ee485)
  - http://www.davidsbatista.net/blog/2018/02/23/model_optimization/ (http://www.davidsbatista.net/blog/2018/02/23/model_optimization/)
  - https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search (https://www.kaggle.com/willkoehrsen/intro-to-model-tuning-grid-and-random-search)
  - https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_grid_search/ (https://chrisalbon.com/machine_learning/model_selection/hyperparameter_tuning_using_grid_search/)
  - https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html (https://jakevdp.github.io/PythonDataScienceHandbook/05.03-hyperparameters-and-model-validation.html)
  - https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35 (https://medium.com/@jackstalfort/hyperparameter-tuning-using-grid-search-and-random-search-f8750a464b35)
  - https://www.kaggle.com/hatone/gradientboostingclassifier-with-gridsearchcv (https://www.kaggle.com/hatone/gradientboostingclassifier-with-gridsearchcv)
  - https://www.datacareer.ch/blog/parameter-tuning-in-gradient-boosting-gbm-with-python/ (https://www.datacareer.ch/blog/parameter-tuning-in-gradient-boosting-gbm-with-python/)
  - https://www.kaggle.com/c/home-credit-default-risk/discussion/60657 (https://www.kaggle.com/c/home-credit-default-risk/discussion/60657)

In [1]:
```
#import warnings
#warnings.filterwarnings('ignore')

#sklearn models
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, \
        GradientBoostingClassifier, StackingClassifier, BaggingClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC, SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.neighbors import KNeighborsClassifier

#sklearn metrics
from sklearn.metrics import roc_curve, auc, average_precision_score, precision_recall_curve
#DOES NOT WORK: , plot_precision_recall_curve
from sklearn.metrics import fbeta_score, make_scorer
from sklearn.metrics import balanced_accuracy_score, precision_score, recall_score, f1_score, roc_auc_
score

#sklearn model selection
from sklearn.model_selection import cross_val_score, cross_validate, train_test_split, cross_val_predi
ct

#scikit-plot
from scikitplot.metrics import plot_precision_recall, plot_roc, plot_confusion_matrix
from scikitplot.estimators import plot_learning_curve

#mlxtend
from mlxtend.plotting import plot_learning_curves, plot_decision_regions


#hyperparameter tuning

#miscellaneous
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

#some "global" variables defined here
random_state_num = 0
cv_num = 10
classes_num = 2 #there are two classes, non-negative (0) and negative (1)
fig_size_tuple = (15,7)
title_fontsize_num = 15
label_fontsize_num = 12
```

Read Trained Word2Vec Embedding Vectors and Engineered Features

We read in the engineered features dataset as a dataframe.

In [2]:
```
#read the csv of engineered features
#then split the columns into
df_gensim_word2vec_features = pd.read_csv('..\data\gensim_word2vec_trained_with_engineered_features.cs
v')
X = df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns != 'binary_response_variabl
e']
Y= df_gensim_word2vec_features.loc[:, df_gensim_word2vec_features.columns == 'binary_response_variabl
e']
```

Split into Train and Test dataset

The dataset is then split into a train and test dataset.

In [3]:
```
#Split into training set, and test set
# we do not need a validation set because we will be doing k-fold cross validation

#split into 0.7 training, and 0.3 testing set
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state = random_state_n
um)

#convert Y_train and Y_test into 1-d array so we don't get stupid warnings about needing a 1d array fo
r Y, in cross_validate function
Y_train_1d = Y_train.values.ravel()
Y_test_1d = Y_test.values.ravel()

#convert Y_train and Y_test into numpy array
Y_train_np_array = Y_train.to_numpy()
Y_test_np_array = Y_test.to_numpy()
```

Models

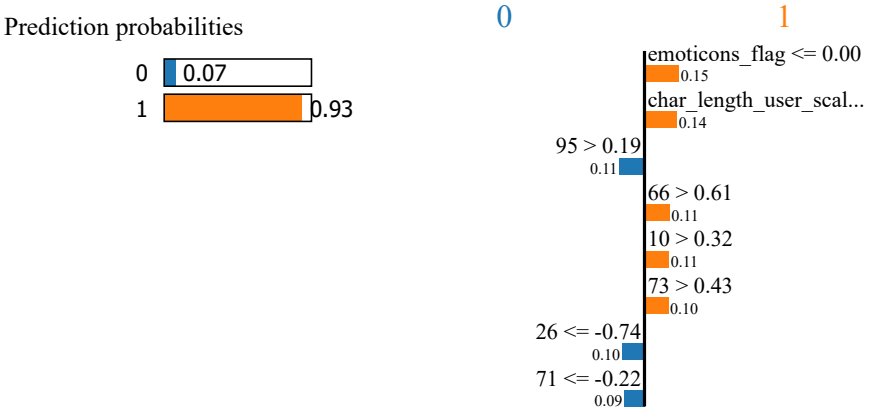
Here we define a set of "basic" models we use as benchmarks to see which model algorithm is better than the others.

In [4]:
```
stacking_base_learners = [
    ('sbl_1', LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, \
                                                 class_weight={1: 0.4, 0: 0.6}, penalty='l1', solver='lib
linear', )),
    ('sbl_2', KNeighborsClassifier(n_neighbors=3, )),
    ('sbl_3', DecisionTreeClassifier()),
    ('sbl_4', GaussianNB())
]

models = [
# Hyperparameter tuned GBM model
#     GradientBoostingClassifier(random_state = random_state_num, min_samples_split=140, min_samples_l
eaf=14, \
#                                subsample=0.9, n_estimators=10, max_features='sqrt', max_depth=8, \
#                                loss='deviance', learning_rate=0.15000000000000002, criterion='friedm
an_mse')

 #   KNeighborsClassifier(n_neighbors=3, ),
  #  DecisionTreeClassifier(),
   # SVC(C = 1000000, gamma = 'auto', kernel = 'rbf', probability=True),
    #GaussianNB(),
    #increased max_iter because it failed to converge at 100
    LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, class_weight={1: 0.4, 0:
0.6}, \
                       penalty='l1', solver='liblinear'),
    #AdaBoostClassifier(random_state = random_state_num),
    #RandomForestClassifier(random_state = random_state_num),
    #GradientBoostingClassifier(random_state = random_state_num),
    #BaggingClassifier(base_estimator=LogisticRegression(random_state = random_state_num, max_iter = 5
00, C=1, \
    #                                                    class_weight={1: 0.4, 0: 0.6}, penalty='l1',
 solver='liblinear')),
    #StackingClassifier(estimators=stacking_base_learners, final_estimator=LogisticRegression())
]
```

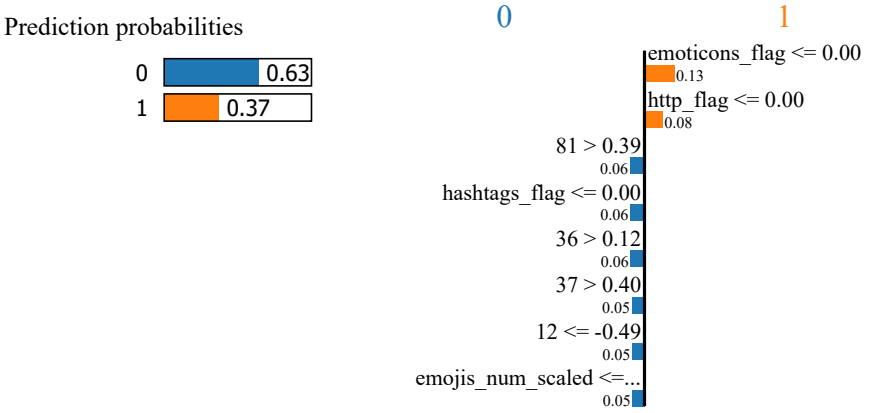
LIME

In [6]:

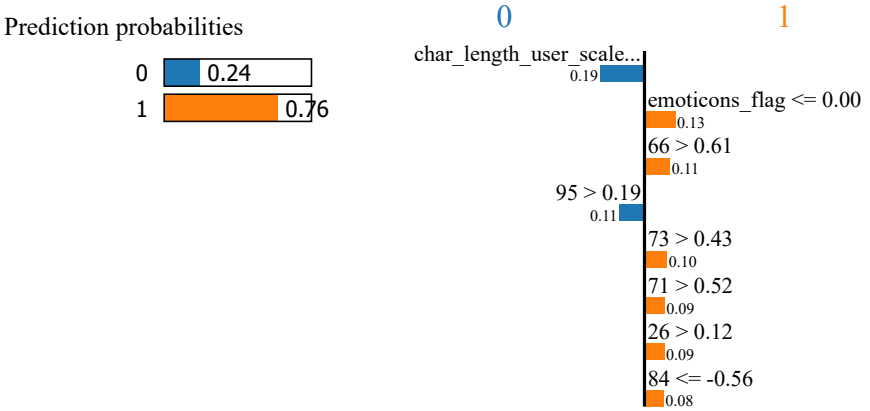
```
logReg_train = Y_train_np_array
logReg_eval = Y_test_np_array

logReg = LogisticRegression(random_state = random_state_num, max_iter = 500, C=1, class_weight={1: 0.4
, 0: 0.6}, \
                        penalty='l1', solver='liblinear')

Y_train_predictions = cross_val_predict(
        logReg,
        X_train,
        Y_train_1d,
        cv = cv_num,
        method = 'predict_proba',
        n_jobs=-1
    )

logReg.fit(X_train, Y_train_1d)

trained_model = logReg.fit(X_train, Y_train_1d)
Y_test_predictions = trained_model.predict_proba(X_test)
Y_test_binary_predictions = trained_model.predict(X_test)

import lime.lime_tabular

explainer = lime.lime_tabular.LimeTabularExplainer(X_train.values,
                feature_names=X_train.columns.values.tolist(),
                class_names=['0' , '1'])

predict_fn = lambda x: logReg.predict_proba(x).astype(float)

exp = explainer.explain_instance(X_test.values[17], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)

exp = explainer.explain_instance(X_test.values[16], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)

exp = explainer.explain_instance(X_test.values[18], predict_fn, num_features=8)
exp.show_in_notebook(show_all=False)
```



Feature Value



Feature Value



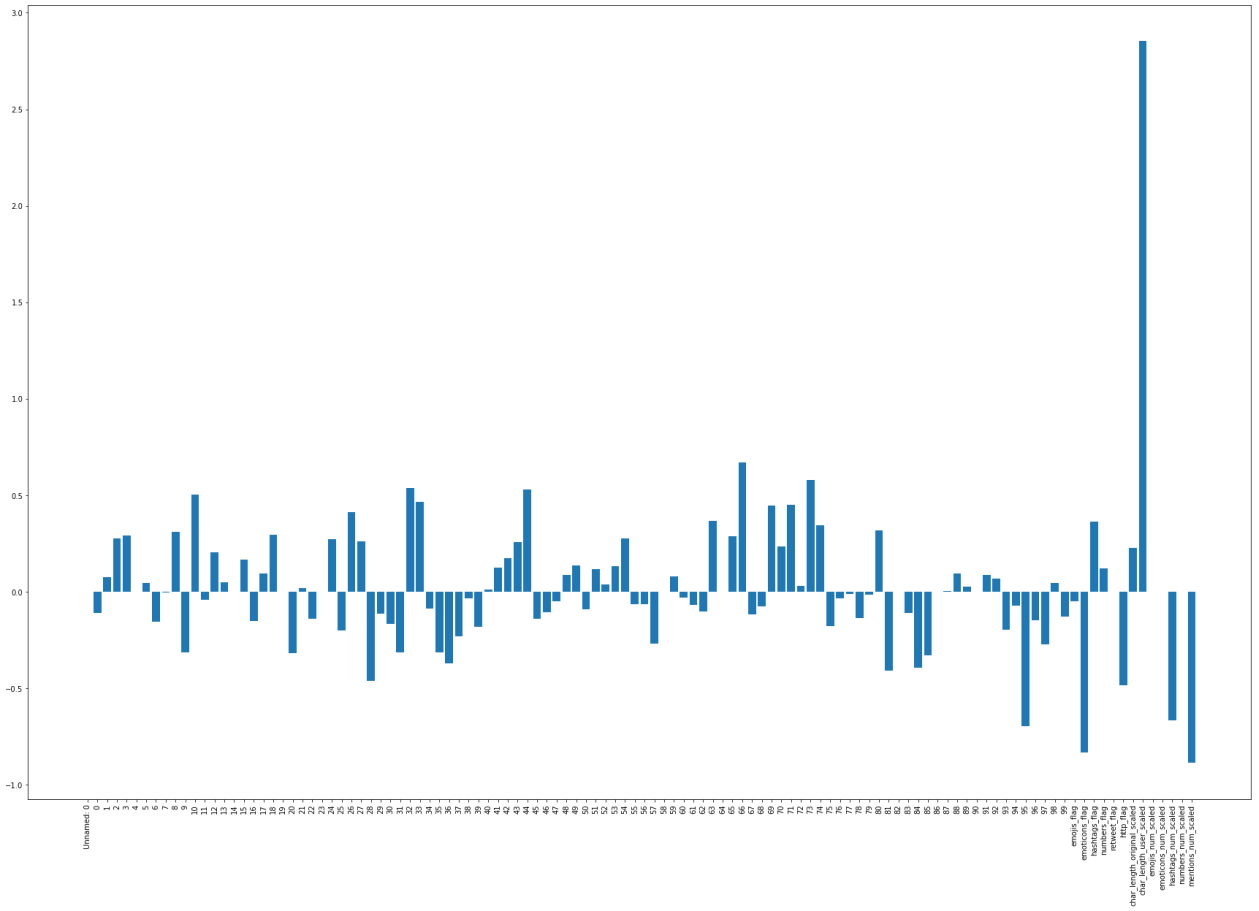
Feature Value

In [36]:

```
feature_names =X_train.columns

importance = logReg.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
        print('Feature: %0s, Score: %.5f' % (feature_names[i],v))
# plot feature importance
plt.figure(figsize=(20,10))
plt.bar([x for x in range(len(importance))], importance)
plt.xticks(range(X_train.shape[1]), feature_names)
plt.xticks(rotation=90)
plt.savefig('logRegFeatureImportance.png')
plt.show()
```

Feature: Unnamed: 0, Score: 0.00002
Feature: 0, Score: -0.10906
Feature: 1, Score: 0.07456
Feature: 2, Score: 0.27539
Feature: 3, Score: 0.29285
Feature: 4, Score: -0.00020
Feature: 5, Score: 0.04547
Feature: 6, Score: -0.15652
Feature: 7, Score: -0.00226
Feature: 8, Score: 0.30894
Feature: 9, Score: -0.31224
Feature: 10, Score: 0.50387
Feature: 11, Score: -0.04224
Feature: 12, Score: 0.20386
Feature: 13, Score: 0.05106
Feature: 14, Score: 0.00000
Feature: 15, Score: 0.16626
Feature: 16, Score: -0.15221
Feature: 17, Score: 0.09515
Feature: 18, Score: 0.29535
Feature: 19, Score: 0.00000
Feature: 20, Score: -0.31937
Feature: 21, Score: 0.02017
Feature: 22, Score: -0.13856
Feature: 23, Score: 0.00000
Feature: 24, Score: 0.27456
Feature: 25, Score: -0.20008
Feature: 26, Score: 0.41356
Feature: 27, Score: 0.26180
Feature: 28, Score: -0.45992
Feature: 29, Score: -0.11188
Feature: 30, Score: -0.16764
Feature: 31, Score: -0.31510
Feature: 32, Score: 0.53725
Feature: 33, Score: 0.46635
Feature: 34, Score: -0.08511
Feature: 35, Score: -0.31413
Feature: 36, Score: -0.36980
Feature: 37, Score: -0.23255
Feature: 38, Score: -0.03237
Feature: 39, Score: -0.17986
Feature: 40, Score: 0.01004
Feature: 41, Score: 0.12500
Feature: 42, Score: 0.17552
Feature: 43, Score: 0.25745
Feature: 44, Score: 0.52913
Feature: 45, Score: -0.14077
Feature: 46, Score: -0.10428
Feature: 47, Score: -0.04785
Feature: 48, Score: 0.08908
Feature: 49, Score: 0.13657
Feature: 50, Score: -0.09150
Feature: 51, Score: 0.11837
Feature: 52, Score: 0.03686
Feature: 53, Score: 0.13400
Feature: 54, Score: 0.27807
Feature: 55, Score: -0.06278
Feature: 56, Score: -0.06548
Feature: 57, Score: -0.26977
Feature: 58, Score: 0.00000
Feature: 59, Score: 0.08153
Feature: 60, Score: -0.03188
Feature: 61, Score: -0.06688
Feature: 62, Score: -0.10128
Feature: 63, Score: 0.36552
Feature: 64, Score: 0.00000
Feature: 65, Score: 0.28738
Feature: 66, Score: 0.67052
Feature: 67, Score: -0.11655
Feature: 68, Score: -0.07475
Feature: 69, Score: 0.44511

Feature: 70, Score: 0.23567
Feature: 71, Score: 0.44879
Feature: 72, Score: 0.03095
Feature: 73, Score: 0.57961
Feature: 74, Score: 0.34332
Feature: 75, Score: -0.17812
Feature: 76, Score: -0.03402
Feature: 77, Score: -0.01037
Feature: 78, Score: -0.13590
Feature: 79, Score: -0.01533
Feature: 80, Score: 0.31629
Feature: 81, Score: -0.40870
Feature: 82, Score: 0.00000
Feature: 83, Score: -0.11097
Feature: 84, Score: -0.39265
Feature: 85, Score: -0.32938
Feature: 86, Score: 0.00000
Feature: 87, Score: 0.00287
Feature: 88, Score: 0.09670
Feature: 89, Score: 0.02624
Feature: 90, Score: 0.00000
Feature: 91, Score: 0.08645
Feature: 92, Score: 0.06919
Feature: 93, Score: -0.19625
Feature: 94, Score: -0.07249
Feature: 95, Score: -0.69517
Feature: 96, Score: -0.14701
Feature: 97, Score: -0.27087
Feature: 98, Score: 0.04453
Feature: 99, Score: -0.12889
Feature: emojis_flag, Score: -0.05040
Feature: emoticons_flag, Score: -0.83323
Feature: hashtags_flag, Score: 0.36318
Feature: numbers_flag, Score: 0.12090
Feature: retweet_flag, Score: 0.00000
Feature: http_flag, Score: -0.48279
Feature: char_length_original_scaled, Score: 0.22784
Feature: char_length_user_scaled, Score: 2.85418
Feature: emojis_num_scaled, Score: 0.00000
Feature: emoticons_num_scaled, Score: 0.00000
Feature: hashtags_num_scaled, Score: -0.66614
Feature: numbers_num_scaled, Score: 0.00000
Feature: mentions_num_scaled, Score: -0.88700



References

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Interpreting Machine Learning Models: https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f (https://towardsdatascience.com/interpretability-in-machine-learning-70c30694a05f)
- Interpreting Machine Learning Models: https://medium.com/ansaro-blog/interpreting-machine-learning-models-1234d735d6c9 (https://medium.com/ansaro-blog/interpreting-machine-learning-models-1234d735d6c9)
- SciKit-Learn API Reference: https://scikit-learn.org/stable/modules/classes.html (https://scikit-learn.org/stable/modules/classes.html)
- Respective API Reference for each package used
- Feature Importance and Model Interpretability (including LIME):
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://www.analyticsvidhya.com/blog/2017/06/building-trust-in-machine-learning-models/ (https://www.analyticsvidhya.com/blog/2017/06/building-trust-in-machine-learning-models/)
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5 (https://towardsdatascience.com/decrypting-your-machine-learning-model-using-lime-5adc035109b5)
  - https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_text (https://lime-ml.readthedocs.io/en/latest/lime.html#module-lime.lime_text)
  - https://marcotcr.github.io/lime/tutorials/Lime%20-%20basic%20usage%2C%20two%20class%20case.html (https://marcotcr.github.io/lime/tutorials/Lime%20-%20basic%20usage%2C%20two%20class%20case.html)
  - https://www.kdnuggets.com/2019/09/python-libraries-interpretable-machine-learning.html (https://www.kdnuggets.com/2019/09/python-libraries-interpretable-machine-learning.html)

In [1]:
```
#BERT code is from https://github.com/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb
```

In [ ]:
```
!pip install bert-tensorflow
```

In [2]:
```
import pandas as pd
from datetime import datetime
import tensorflow as tf
import tensorflow_hub as hub
from sklearn.model_selection import train_test_split

import bert
from bert import run_classifier
from bert import optimization
from bert import tokenization
```

```
WARNING:tensorflow:From C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\bert
\optimization.py:87: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimiz
er instead.
```

In [3]:
```
df_tweets_cleaned = pd.read_csv('..\data\Tweets_cleaned.csv')
```

In [4]:
```
df_tweets_cleaned
```

Out[4]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airlir
0	570306133677760513	neutral	1.0000	NaN	NaN	Virg Americ
1	570301130888122368	positive	0.3486	NaN	0.0000	Virg Americ
2	570301083672813571	neutral	0.6837	NaN	NaN	Virg Americ
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virg Americ
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virg Americ
...	...	...	...	...	...	
14635	569587686496825344	positive	0.3487	NaN	0.0000	Americ
14636	569587371693355008	negative	1.0000	Customer Service Issue	1.0000	Americ
14637	569587242672398336	neutral	1.0000	NaN	NaN	Americ
14638	569587188687634433	negative	1.0000	Customer Service Issue	0.6659	Americ
14639	569587140490866689	neutral	0.6771	NaN	0.0000	Americ

14640 rows × 21 columns

In [5]:
```
#df_tweets_bert = df_tweets_cleaned[['airline_sentiment', 'text_cleaned']]
df_tweets_bert = df_tweets_cleaned[['airline_sentiment', 'text_list_no_stop_words']]
```

In [6]:
```
df_tweets_bert
```

Out[6]:

	airline_sentiment	text_list_no_stop_words
0	neutral	said
1	positive	plus added commercials experience tacky
2	neutral	today mean need trip
3	negative	aggressive blast obnoxious entertainment guest...
4	negative	big bad thing
...	...	...
14635	positive	thank got different flight chicago
14636	negative	leaving minutes late flight warnings communica...
14637	neutral	bring american airlines
14638	negative	money change flight answer phones suggestions ...
14639	neutral	people need know seats flight standby people f...

14640 rows × 2 columns

In [7]:
```
df_tweets_bert['binary_response_variable'] = False

df_tweets_bert.loc[df_tweets_cleaned.airline_sentiment == 'neutral', 'binary_response_variable'] = Fal
se
df_tweets_bert.loc[df_tweets_cleaned.airline_sentiment == 'positive', 'binary_response_variable'] = Fa
lse
df_tweets_bert.loc[df_tweets_cleaned.airline_sentiment == 'negative', 'binary_response_variable'] = Tr
ue
```

```
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\ipykernel_launcher.py:1: Sett
ingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\pandas\core\indexing.py:966:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  self.obj[item] = s
```

In [8]:
```
df_tweets_bert.binary_response_variable = df_tweets_bert.binary_response_variable.astype(int)
```

```
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\pandas\core\generic.py:5303:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexin
g.html#returning-a-view-versus-a-copy
  self[name] = value
```

In [9]:
```
df_tweets_bert = df_tweets_bert[['text_list_no_stop_words', 'binary_response_variable']]
df_tweets_bert
```

Out[9]:

	text_list_no_stop_words	binary_response_variable
0	said	0
1	plus added commercials experience tacky	0
2	today mean need trip	0
3	aggressive blast obnoxious entertainment guest...	1
4	big bad thing	1
...	...	...
14635	thank got different flight chicago	0
14636	leaving minutes late flight warnings communica...	1
14637	bring american airlines	0
14638	money change flight answer phones suggestions ...	1
14639	people need know seats flight standby people f...	0

14640 rows × 2 columns

In [10]:
```
X = df_tweets_bert.loc[:, df_tweets_bert.columns != 'binary_response_variable']
Y = df_tweets_bert.loc[:, df_tweets_bert.columns == 'binary_response_variable']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1, stratify=Y)
```

In [11]:
```
X_train
```

Out[11]:

	text_list_no_stop_words
4771	arrangements reimburse rental
12454	apologizing rude sales reps failure offer trit...
3840	year old flying tokyo vacation bad knees happens
14176	baggage lost flight cancelled flightled accomm...
1219	passengers rerouted match intended arrival tim...
...	...
7233	yeah aware original message
3855	free upgrade problems reservation mkwlkr
6339	hey guys honest customers unlike
11606	jfk baggage office open help book cancelled fl...
3899	pri boarding active military uniform travel un...

10248 rows × 1 columns

In [12]:
```
XY_train = pd.concat([X_train, Y_train], axis=1)
XY_test = pd.concat([X_test, Y_test], axis=1)
```

In [13]:
```
XY_train = XY_train.dropna()
XY_test = XY_test.dropna()
```

In [14]:
```
print(XY_train.isnull().values.any())
print(XY_test.isnull().values.any())
```

```
False
False
```

In [15]:
```
DATA_COLUMN = 'text_list_no_stop_words'
LABEL_COLUMN = 'binary_response_variable'

# label_list is the list of labels. In this situation they are 0, 1
label_list = [0, 1]
```

In [16]:
```
XY_train
```

Out[16]:

	text_list_no_stop_words	binary_response_variable
4771	arrangements reimburse rental	0
12454	apologizing rude sales reps failure offer trit...	1
3840	year old flying tokyo vacation bad knees happens	1
14176	baggage lost flight cancelled flightled accomm...	1
1219	passengers rerouted match intended arrival tim...	1
...	...	...
7233	yeah aware original message	1
3855	free upgrade problems reservation mkwlkr	1
6339	hey guys honest customers unlike	0
11606	jfk baggage office open help book cancelled fl...	1
3899	pri boarding active military uniform travel un...	0

10190 rows × 2 columns

Data Preprocessing

In [17]:
```
# Use the InputExample class from BERT's run_classifier code to create examples from the data
train_InputExamples = XY_train.apply(lambda x: bert.run_classifier.InputExample(guid=None, # Globally
 unique ID for bookkeeping, unused in this example
                                                                                 text_a = x[DATA_COLUMN],
                                                                                 text_b = None,
                                                                                 label = x[LABEL_COLUMN]), axis = 1)

test_InputExamples = XY_test.apply(lambda x: bert.run_classifier.InputExample(guid=None,
                                                                                 text_a = x[DATA_COLUMN],
                                                                                 text_b = None,
                                                                                 label = x[LABEL_COLUMN]), axis = 1)
```

In [18]:
```
train_InputExamples
```

Out[18]:
```
4771     <bert.run_classifier.InputExample object at 0x...
12454    <bert.run_classifier.InputExample object at 0x...
3840     <bert.run_classifier.InputExample object at 0x...
14176    <bert.run_classifier.InputExample object at 0x...
1219     <bert.run_classifier.InputExample object at 0x...
                               ...
7233     <bert.run_classifier.InputExample object at 0x...
3855     <bert.run_classifier.InputExample object at 0x...
6339     <bert.run_classifier.InputExample object at 0x...
11606    <bert.run_classifier.InputExample object at 0x...
3899     <bert.run_classifier.InputExample object at 0x...
Length: 10190, dtype: object
```

In [19]:
```
# This is a path to an uncased (all lowercase) version of BERT
BERT_MODEL_HUB = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"

def create_tokenizer_from_hub_module():
  """Get the vocab file and casing info from the Hub module."""
  with tf.Graph().as_default():
    bert_module = hub.Module(BERT_MODEL_HUB)
    tokenization_info = bert_module(signature="tokenization_info", as_dict=True)
    with tf.Session() as sess:
      vocab_file, do_lower_case = sess.run([tokenization_info["vocab_file"],
                                            tokenization_info["do_lower_case"]])

  return bert.tokenization.FullTokenizer(
      vocab_file=vocab_file, do_lower_case=do_lower_case)

tokenizer = create_tokenizer_from_hub_module()
```

```
INFO:tensorflow:Saver not created because there are no variables in the graph to restore

INFO:tensorflow:Saver not created because there are no variables in the graph to restore

WARNING:tensorflow:From C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\bert
\tokenization.py:125: The name tf.gfile.GFile is deprecated. Please use tf.io.gfile.GFile instead.


WARNING:tensorflow:From C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\bert
\tokenization.py:125: The name tf.gfile.GFile is deprecated. Please use tf.io.gfile.GFile instead.
```

In [20]:
```
# We'll set sequences to be at most 128 tokens long.
MAX_SEQ_LENGTH = 128
# Convert our train and test features to InputFeatures that BERT understands.
train_features = bert.run_classifier.convert_examples_to_features(train_InputExamples, label_list, MAX
_SEQ_LENGTH, tokenizer)
test_features = bert.run_classifier.convert_examples_to_features(test_InputExamples, label_list, MAX_S
EQ_LENGTH, tokenizer)
```

WARNING:tensorflow:From C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\bert
\run_classifier.py:774: The name tf.logging.info is deprecated. Please use tf.compat.v1.logging.info
instead.

WARNING:tensorflow:From C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\bert
\run_classifier.py:774: The name tf.logging.info is deprecated. Please use tf.compat.v1.logging.info
instead.

INFO:tensorflow:Writing example 0 of 10190

INFO:tensorflow:Writing example 0 of 10190

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] arrangements rei ##mb ##urse rental [SEP]

INFO:tensorflow:tokens: [CLS] arrangements rei ##mb ##urse rental [SEP]

INFO:tensorflow:input_ids: 101 7565 24964 14905 28393 12635 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 7565 24964 14905 28393 12635 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] ap ##olo ##gizing rude sales rep ##s failure offer tri ##te conde ##sc
##ending pl ##ati ##tu ##des [SEP]

INFO:tensorflow:tokens: [CLS] ap ##olo ##gizing rude sales rep ##s failure offer tri ##te conde ##sc
##ending pl ##ati ##tu ##des [SEP]

INFO:tensorflow:input_ids: 101 9706 12898 28660 12726 4341 16360 2015 4945 3749 13012 2618 24707 1102
0 18537 20228 10450 8525 6155 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
INFO:tensorflow:input_ids: 101 9706 12898 28660 12726 4341 16360 2015 4945 3749 13012 2618 24707 1102
0 18537 20228 10450 8525 6155 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:label: 1 (id = 1)
```


```
INFO:tensorflow:label: 1 (id = 1)
```


```
INFO:tensorflow:*** Example ***
```


```
INFO:tensorflow:*** Example ***
```


```
INFO:tensorflow:guid: None
```


```
INFO:tensorflow:guid: None
```


```
INFO:tensorflow:tokens: [CLS] year old flying tokyo vacation bad knees happens [SEP]
```


```
INFO:tensorflow:tokens: [CLS] year old flying tokyo vacation bad knees happens [SEP]
```


```
INFO:tensorflow:input_ids: 101 2095 2214 3909 5522 10885 2919 5042 6433 102 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
```


```
INFO:tensorflow:input_ids: 101 2095 2214 3909 5522 10885 2919 5042 6433 102 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0
```


```
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```


```
INFO:tensorflow:label: 1 (id = 1)
```


```
INFO:tensorflow:label: 1 (id = 1)
```


```
INFO:tensorflow:*** Example ***
```


```
INFO:tensorflow:*** Example ***
```


```
INFO:tensorflow:guid: None
```


```
INFO:tensorflow:guid: None
```

INFO:tensorflow:tokens: [CLS] baggage lost flight cancelled flight ##led accommodations airline free hotel [SEP]

INFO:tensorflow:tokens: [CLS] baggage lost flight cancelled flight ##led accommodations airline free hotel [SEP]

INFO:tensorflow:input_ids: 101 20220 2439 3462 8014 3462 3709 26167 8582 2489 3309 102 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 20220 2439 3462 8014 3462 3709 26167 8582 2489 3309 102 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] passengers re ##routed match intended arrival time inc ##ur cost airlin
es [SEP]

INFO:tensorflow:tokens: [CLS] passengers re ##routed match intended arrival time inc ##ur cost airlin
es [SEP]

INFO:tensorflow:input_ids: 101 5467 2128 25849 2674 3832 5508 2051 4297 3126 3465 7608 102 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 5467 2128 25849 2674 3832 5508 2051 4297 3126 3465 7608 102 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```
INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:Writing example 10000 of 10190

INFO:tensorflow:Writing example 10000 of 10190

INFO:tensorflow:Writing example 0 of 4370

INFO:tensorflow:Writing example 0 of 4370

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] hold flight sl ##c den [SEP]

INFO:tensorflow:tokens: [CLS] hold flight sl ##c den [SEP]

INFO:tensorflow:input_ids: 101 2907 3462 22889 2278 7939 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 2907 3462 22889 2278 7939 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] add wi ##fi entertainment old seats come [SEP]

INFO:tensorflow:tokens: [CLS] add wi ##fi entertainment old seats come [SEP]

INFO:tensorflow:input_ids: 101 5587 15536 8873 4024 2214 4272 2272 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

INFO:tensorflow:input_ids: 101 5587 15536 8873 4024 2214 4272 2272 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] change united inc ##omp ##ete ##nce flight monday hour drive flight [SE
P]

INFO:tensorflow:tokens: [CLS] change united inc ##omp ##ete ##nce flight monday hour drive flight [SE
P]

INFO:tensorflow:input_ids: 101 2689 2142 4297 25377 12870 5897 3462 6928 3178 3298 3462 102 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 2689 2142 4297 25377 12870 5897 3462 6928 3178 3298 3462 102 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] thanks understand bags fit notified air landing [SEP]

INFO:tensorflow:tokens: [CLS] thanks understand bags fit notified air landing [SEP]

INFO:tensorflow:input_ids: 101 4283 3305 8641 4906 19488 2250 4899 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

INFO:tensorflow:input_ids: 101 4283 3305 8641 4906 19488 2250 4899 102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:label: 1 (id = 1)

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid: None

INFO:tensorflow:guid: None

INFO:tensorflow:tokens: [CLS] thanks getting nashville big thanks pilots 6 : 15 baltimore flying snow landing ice [SEP]

INFO:tensorflow:tokens: [CLS] thanks getting nashville big thanks pilots 6 : 15 baltimore flying snow landing ice [SEP]

INFO:tensorflow:input_ids: 101 4283 2893 8423 2502 4283 8221 1020 1024 2321 6222 3909 4586 4899 3256
102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_ids: 101 4283 2893 8423 2502 4283 8221 1020 1024 2321 6222 3909 4586 4899 3256
102 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

In [21]:
```
train_InputExamples
```

Out[21]:
```
4771     <bert.run_classifier.InputExample object at 0x...
12454    <bert.run_classifier.InputExample object at 0x...
3840     <bert.run_classifier.InputExample object at 0x...
14176    <bert.run_classifier.InputExample object at 0x...
1219     <bert.run_classifier.InputExample object at 0x...
                                  ...
7233     <bert.run_classifier.InputExample object at 0x...
3855     <bert.run_classifier.InputExample object at 0x...
6339     <bert.run_classifier.InputExample object at 0x...
11606    <bert.run_classifier.InputExample object at 0x...
3899     <bert.run_classifier.InputExample object at 0x...
Length: 10190, dtype: object
```

Model

In [22]:

```
def create_model(is_predicting, input_ids, input_mask, segment_ids, labels,
                 num_labels):
  """Creates a classification model."""

  bert_module = hub.Module(
      BERT_MODEL_HUB,
      trainable=True)
  bert_inputs = dict(
      input_ids=input_ids,
      input_mask=input_mask,
      segment_ids=segment_ids)
  bert_outputs = bert_module(
      inputs=bert_inputs,
      signature="tokens",
      as_dict=True)

  # Use "pooled_output" for classification tasks on an entire sentence.
  # Use "sequence_outputs" for token-level output.
  output_layer = bert_outputs["pooled_output"]

  hidden_size = output_layer.shape[-1].value

  # Create our own layer to tune for politeness data.
  output_weights = tf.get_variable(
      "output_weights", [num_labels, hidden_size],
      initializer=tf.truncated_normal_initializer(stddev=0.02))

  output_bias = tf.get_variable(
      "output_bias", [num_labels], initializer=tf.zeros_initializer())

  with tf.variable_scope("loss"):

    # Dropout helps prevent overfitting
    output_layer = tf.nn.dropout(output_layer, keep_prob=0.9)

    logits = tf.matmul(output_layer, output_weights, transpose_b=True)
    logits = tf.nn.bias_add(logits, output_bias)
    log_probs = tf.nn.log_softmax(logits, axis=-1)

    # Convert labels into one-hot encoding
    one_hot_labels = tf.one_hot(labels, depth=num_labels, dtype=tf.float32)

    predicted_labels = tf.squeeze(tf.argmax(log_probs, axis=-1, output_type=tf.int32))
    # If we're predicting, we want predicted labels and the probabilties.
    if is_predicting:
      return (predicted_labels, log_probs)

    # If we're train/eval, compute loss between predicted and actual label
    per_example_loss = -tf.reduce_sum(one_hot_labels * log_probs, axis=-1)
    loss = tf.reduce_mean(per_example_loss)
    return (loss, predicted_labels, log_probs)
```

In [23]:

```
# model_fn_builder actually creates our model function
# using the passed parameters for num_labels, learning_rate, etc.
def model_fn_builder(num_labels, learning_rate, num_train_steps,
                     num_warmup_steps):
  """Returns `model_fn` closure for TPUEstimator."""
  def model_fn(features, labels, mode, params):  # pylint: disable=unused-argument
    """The `model_fn` for TPUEstimator."""

    input_ids = features["input_ids"]
    input_mask = features["input_mask"]
    segment_ids = features["segment_ids"]
    label_ids = features["label_ids"]

    is_predicting = (mode == tf.estimator.ModeKeys.PREDICT)

    # TRAIN and EVAL
    if not is_predicting:

      (loss, predicted_labels, log_probs) = create_model(
        is_predicting, input_ids, input_mask, segment_ids, label_ids, num_labels)

      train_op = bert.optimization.create_optimizer(
          loss, learning_rate, num_train_steps, num_warmup_steps, use_tpu=False)

      # Calculate evaluation metrics.
      def metric_fn(label_ids, predicted_labels):
        accuracy = tf.metrics.accuracy(label_ids, predicted_labels)
        f1_score = tf.contrib.metrics.f1_score(
            label_ids,
            predicted_labels)
#         f2_score = tf.contrib.metrics.f2_score(
#             label_ids,
#             predicted_labels
#         )
        auc = tf.metrics.auc(
            label_ids,
            predicted_labels)
        recall = tf.metrics.recall(
            label_ids,
            predicted_labels)
        precision = tf.metrics.precision(
            label_ids,
            predicted_labels)
        true_pos = tf.metrics.true_positives(
            label_ids,
            predicted_labels)
        true_neg = tf.metrics.true_negatives(
            label_ids,
            predicted_labels)
        false_pos = tf.metrics.false_positives(
            label_ids,
            predicted_labels)
        false_neg = tf.metrics.false_negatives(
            label_ids,
            predicted_labels)
        return {
            "eval_accuracy": accuracy,
            "f1_score": f1_score,
#             "f2_score": f2_score,
            "auc": auc,
            "precision": precision,
            "recall": recall,
            "true_positives": true_pos,
            "true_negatives": true_neg,
            "false_positives": false_pos,
            "false_negatives": false_neg
        }

      eval_metrics = metric_fn(label_ids, predicted_labels)

      if mode == tf.estimator.ModeKeys.TRAIN:
```

```
        return tf.estimator.EstimatorSpec(mode=mode,
          loss=loss,
          train_op=train_op)
      else:
          return tf.estimator.EstimatorSpec(mode=mode,
            loss=loss,
            eval_metric_ops=eval_metrics)
    else:
      (predicted_labels, log_probs) = create_model(
        is_predicting, input_ids, input_mask, segment_ids, label_ids, num_labels)

      predictions = {
          'probabilities': log_probs,
          'labels': predicted_labels
      }
      return tf.estimator.EstimatorSpec(mode, predictions=predictions)

  # Return the actual model function in the closure
  return model_fn
```

In [24]:
```
BATCH_SIZE = 1
LEARNING_RATE = 2e-5
NUM_TRAIN_EPOCHS = 3.0

WARMUP_PROPORTION = 0.1

SAVE_CHECKPOINTS_STEPS = 500
SAVE_SUMMARY_STEPS = 100
```

In [25]:
```
# Compute # train and warmup steps from batch size
num_train_steps = int(len(train_features) / BATCH_SIZE * NUM_TRAIN_EPOCHS)
num_warmup_steps = int(num_train_steps * WARMUP_PROPORTION)
```

In [26]:
```
OUTPUT_DIR = 'bert'

# Specify outpit directory and number of checkpoint steps to save
run_config = tf.estimator.RunConfig(
    model_dir=OUTPUT_DIR,
    save_summary_steps=SAVE_SUMMARY_STEPS,
    save_checkpoints_steps=SAVE_CHECKPOINTS_STEPS)
```

In [27]:
```
model_fn = model_fn_builder(
  num_labels=len(label_list),
  learning_rate=LEARNING_RATE,
  num_train_steps=num_train_steps,
  num_warmup_steps=num_warmup_steps)

estimator = tf.estimator.Estimator(
  model_fn=model_fn,
  config=run_config,
  params={"batch_size": BATCH_SIZE})
```

```
INFO:tensorflow:Using config: {'_model_dir': 'bert', '_tf_random_seed': None, '_save_summary_steps':
100, '_save_checkpoints_steps': 500, '_save_checkpoints_secs': None, '_session_config': allow_soft_pl
acement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experim
ental_distribute': None, '_experimental_max_worker_delay_secs': None, '_session_creation_timeout_sec
s': 7200, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec objec
t at 0x00000272098F33C8>, '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_maste
r': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas':
1}

INFO:tensorflow:Using config: {'_model_dir': 'bert', '_tf_random_seed': None, '_save_summary_steps':
100, '_save_checkpoints_steps': 500, '_save_checkpoints_secs': None, '_session_config': allow_soft_pl
acement: true
graph_options {
  rewrite_options {
    meta_optimizer_iterations: ONE
  }
}
, '_keep_checkpoint_max': 5, '_keep_checkpoint_every_n_hours': 10000, '_log_step_count_steps': 100,
'_train_distribute': None, '_device_fn': None, '_protocol': None, '_eval_distribute': None, '_experim
ental_distribute': None, '_experimental_max_worker_delay_secs': None, '_session_creation_timeout_sec
s': 7200, '_service': None, '_cluster_spec': <tensorflow.python.training.server_lib.ClusterSpec objec
t at 0x00000272098F33C8>, '_task_type': 'worker', '_task_id': 0, '_global_id_in_cluster': 0, '_maste
r': '', '_evaluation_master': '', '_is_chief': True, '_num_ps_replicas': 0, '_num_worker_replicas':
1}
```

In [28]:
```
# Create an input function for training. drop_remainder = True for using TPUs.
train_input_fn = bert.run_classifier.input_fn_builder(
    features=train_features,
    seq_length=MAX_SEQ_LENGTH,
    is_training=True,
    drop_remainder=False)
```

In [29]:
```
print(f'Beginning Training!')
current_time = datetime.now()
estimator.train(input_fn=train_input_fn, max_steps=num_train_steps)
print("Training took time ", datetime.now() - current_time)
```

```
Beginning Training!
INFO:tensorflow:Skipping training since max_steps has already saved.

INFO:tensorflow:Skipping training since max_steps has already saved.

Training took time  0:00:00.048015
```

In [30]:
```
test_input_fn = run_classifier.input_fn_builder(
    features=test_features,
    seq_length=MAX_SEQ_LENGTH,
    is_training=False,
    drop_remainder=False)
```

In [43]:
```
#print(f'Beginning predictions!')
#current_time = datetime.now()
estimator.evaluate(input_fn=test_input_fn, steps=None)
#print("Training predictions took this amount of time ", datetime.now() - current_time)
```

```
INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to restore

INFO:tensorflow:Saver not created because there are no variables in the graph to restore
C:\Users\alexf\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\tensorflow_core\python\framew
ork\indexed_slices.py:424: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown
shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Starting evaluation at 2020-05-22T09:02:40Z

INFO:tensorflow:Starting evaluation at 2020-05-22T09:02:40Z

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from bert\model.ckpt-30570

INFO:tensorflow:Restoring parameters from bert\model.ckpt-30570

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Finished evaluation at 2020-05-22-09:03:53

INFO:tensorflow:Finished evaluation at 2020-05-22-09:03:53

INFO:tensorflow:Saving dict for global step 30570: auc = 0.81412077, eval_accuracy = 0.83272314, f1_s
core = 0.86944085, false_negatives = 311.0, false_positives = 420.0, global_step = 30570, loss = 1.18
03308, precision = 0.8528381, recall = 0.8867031, true_negatives = 1205.0, true_positives = 2434.0

INFO:tensorflow:Saving dict for global step 30570: auc = 0.81412077, eval_accuracy = 0.83272314, f1_s
core = 0.86944085, false_negatives = 311.0, false_positives = 420.0, global_step = 30570, loss = 1.18
03308, precision = 0.8528381, recall = 0.8867031, true_negatives = 1205.0, true_positives = 2434.0

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 30570: bert\model.ckpt-30570

INFO:tensorflow:Saving 'checkpoint_path' summary for global step 30570: bert\model.ckpt-30570
```

Out[43]:
```
{'auc': 0.81412077,
 'eval_accuracy': 0.83272314,
 'f1_score': 0.86944085,
 'false_negatives': 311.0,
 'false_positives': 420.0,
 'loss': 1.1803308,
 'precision': 0.8528381,
 'recall': 0.8867031,
 'true_negatives': 1205.0,
 'true_positives': 2434.0,
 'global_step': 30570}
```

In [37]:

```
def getPrediction(in_sentences):
  labels = [0, 1]
  input_examples = [run_classifier.InputExample(guid="", text_a = x, text_b = None, label = 0) for x i
n in_sentences] # here, "" is just a dummy label
  input_features = run_classifier.convert_examples_to_features(input_examples, label_list, MAX_SEQ_LEN
GTH, tokenizer)
  predict_input_fn = run_classifier.input_fn_builder(features=input_features, seq_length=MAX_SEQ_LENGT
H, is_training=False, drop_remainder=False)
  predictions = estimator.predict(predict_input_fn)
  return [(sentence, prediction['probabilities'], labels[prediction['labels']]) for sentence, predicti
on in zip(in_sentences, predictions)]
```

In [41]:
```
#pred_sentences = [
#  "My flight to New York was delayedn AGAIN"#,
#  #"So excited to be on the flight to San Francisco!"
#]

#predictions = getPrediction(pred_sentences)
#predictions
```

```
INFO:tensorflow:Writing example 0 of 1

INFO:tensorflow:Writing example 0 of 1

INFO:tensorflow:*** Example ***

INFO:tensorflow:*** Example ***

INFO:tensorflow:guid:

INFO:tensorflow:guid:

INFO:tensorflow:tokens: [CLS] my flight to new york was delayed ##n again [SEP]

INFO:tensorflow:tokens: [CLS] my flight to new york was delayed ##n again [SEP]

INFO:tensorflow:input_ids: 101 2026 3462 2000 2047 2259 2001 8394 2078 2153 102 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0

INFO:tensorflow:input_ids: 101 2026 3462 2000 2047 2259 2001 8394 2078 2153 102 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:input_mask: 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:segment_ids: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:label: 0 (id = 0)

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Calling model_fn.

INFO:tensorflow:Saver not created because there are no variables in the graph to restore

INFO:tensorflow:Saver not created because there are no variables in the graph to restore

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Done calling model_fn.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Graph was finalized.

INFO:tensorflow:Restoring parameters from bert\model.ckpt-30570

INFO:tensorflow:Restoring parameters from bert\model.ckpt-30570

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Running local_init_op.

INFO:tensorflow:Done running local_init_op.

INFO:tensorflow:Done running local_init_op.
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-41-2848935e4500> in <module>
----> 1 predictions = getPrediction(pred_sentences)
      2 predictions

<ipython-input-37-0fd105d73268> in getPrediction(in_sentences)
      5   predict_input_fn = run_classifier.input_fn_builder(features=input_features, seq_length=MAX_
SEQ_LENGTH, is_training=False, drop_remainder=False)
      6   predictions = estimator.predict(predict_input_fn)
----> 7   return [(sentence, prediction['probabilities'], labels[prediction['labels']]) for sentence,
prediction in zip(in_sentences, predictions)]

<ipython-input-37-0fd105d73268> in <listcomp>(.0)
      5   predict_input_fn = run_classifier.input_fn_builder(features=input_features, seq_length=MAX_
SEQ_LENGTH, is_training=False, drop_remainder=False)
      6   predictions = estimator.predict(predict_input_fn)
----> 7   return [(sentence, prediction['probabilities'], labels[prediction['labels']]) for sentence,
prediction in zip(in_sentences, predictions)]

~\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\tensorflow_estimator\python\estimator\esti
mator.py in predict(self, input_fn, predict_keys, hooks, checkpoint_path, yield_single_examples)
    645                 yield pred
    646             else:
--> 647               for i in range(self._extract_batch_length(preds_evaluated)):
    648                 yield {
    649                     key: value[i]

~\Anaconda3\envs\CSML1010_OnlineSession2\lib\site-packages\tensorflow_estimator\python\estimator\esti
mator.py in _extract_batch_length(self, preds_evaluated)
   1030     for key, value in six.iteritems(preds_evaluated):
   1031       batch_length = batch_length or value.shape[0]
-> 1032       if value.shape[0] != batch_length:
   1033         raise ValueError('Batch length of predictions should be same. %s has '
   1034                          'different batch length than others.' % key)

IndexError: tuple index out of range
```

References

Code for this project is either directly from (with some modification), or inspired by, but not limited to the following sources:

- Predicting Movie Review Sentiment with BERT on TF Hub: https://github.com/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb (https://github.com/google-research/bert/blob/master/predicting_movie_reviews_with_bert_on_tf_hub.ipynb)
- Introducing BERT with Tensorflow: https://www.kaggle.com/sergeykalutsky/introducing-bert-with-tensorflow (https://www.kaggle.com/sergeykalutsky/introducing-bert-with-tensorflow)

In [ ]:

Final References:

If not otherwise specified, code is either original or has been reproduced with appropriate modification and citation from the provided Coding Exercises and linked articles/resources in the York University CSML1010, Winter 2020 course hosted here: https://learn.continue.yorku.ca/course/view.php?id=3819