US Airlines Twitter Sentiment Analysis

Dasun Wellawalage
DSC 680
https://github.com/dasun27/Bellevue/blob/gh-pages/index.md

Business Problem

User feedback is quite important in any business and specially, in online businesses. Simply because other users read these feedbacks before deciding to use your product or service. Social media platforms like Twitter play a major role in advertising today. Thus, it is quite important to make sure the overall sentiment for your products and services remain positive on these platforms. Passengers had a negative sentiment about most US airlines for quite some time. However, due to some recent events related to passenger safety and customer service, this negative sentiment has increased. I am trying to build a model to predict the user sentiment based on their Twitter comments, which can then be used to analyze find ways to improve the service.

Method

Based on the dataset there are a few methods I can use to analyze this. There is a specific column that has already identified each sentiment as positive, negative, or neutral and there is confidence rating associated too. Initially I was planning to do some regression analysis based on other features like user's location, time zone etc. But most of those features had a lot of null values. Hence, I decided to base my analysis on text field alone at which point it became a text analysis project.

1. Loading Data – This was easy as there was only one data file which was a CSV.

```
data = pd.read_csv('Tweets.csv')

data.head()
```

2. Data Cleanup – There were a few columns with a lot of null values. I dropped some of those and replaced the rest with arbitrary values.

```
data.isnull().sum()
```

```
tweet_id                          0
airline_sentiment                 0
airline_sentiment_confidence      0
negativereason                 5462
negativereason_confidence      4118
airline                           0
airline_sentiment_gold        14600
name                              0
negativereason_gold           14608
retweet_count                     0
text                              0
tweet_coord                   13621
tweet_created                     0
tweet_location                 4733
user_timezone                  4820
dtype: int64
```

3. Drop & Replace – I dropped some columns that didn't have much values and replaced the null values in others. Most sentiments had a confidence rating of 1 while others had something close to that. I chose to omit the sentiments with a confidence rating below 0.5 arbitrarily.

```
data.drop(['airline_sentiment_gold', 'negativereason_gold', 'tweet_coord'], axis=1, inplace=True)

# Replacing NaNs in 'negativereason_confidence' with 0
data['negativereason_confidence'] = data['negativereason_confidence'].fillna(0)

# Removing records with less than 0.5 sentiment confidence rating
ds = data[data['airline_sentiment_confidence'] > 0.5]

# Replacing remaining null values with arbitrary values
ds['negativereason'] = ds['negativereason'].fillna('NoReason')
ds['tweet_location'] = ds['tweet_location'].fillna('NoLocation')
ds['user_timezone'] = ds['user_timezone'].fillna('NoTimezone')

# Removing 'retweet_count' column as well as it is mostly zeros.
ds.drop(['retweet_count'], axis=1, inplace=True)
```

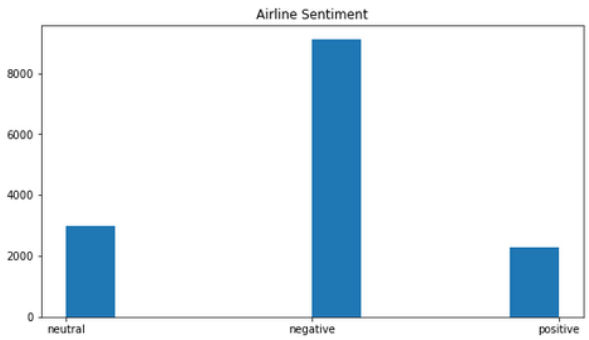
4. Data Exploration – Here, I used some visualization techniques to explore this dataset. First, I checked for unique values in each column. Based on that ‘airline’ and ‘negativereason’ seemed good for visualizations.

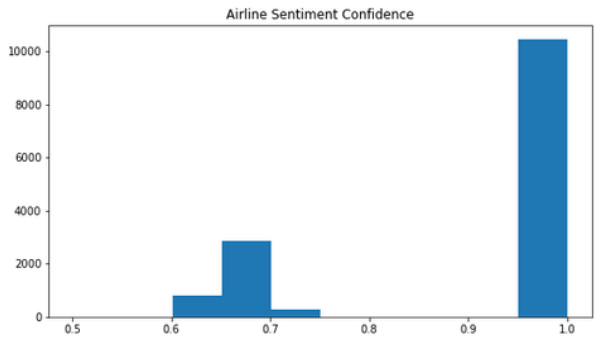
```
for x in list(ds.columns):
    print(x, ":", len(ds[x].unique()))temp_Age = df[df['Age'] < 80]
```

```
tweet_id : 2424
airline_sentiment : 3
airline_sentiment_confidence : 868
negativereason : 11
negativereason_confidence : 1405
airline : 6
name : 7624
text : 14197
tweet_created : 6750
tweet_location : 3054
user_timezone : 85
```

5. Visualizing airline sentiments - As you can see in the histogram below, a vast majority of sentiments about US airlines were negative. You will see how this affects the model accuracy later in the analysis.

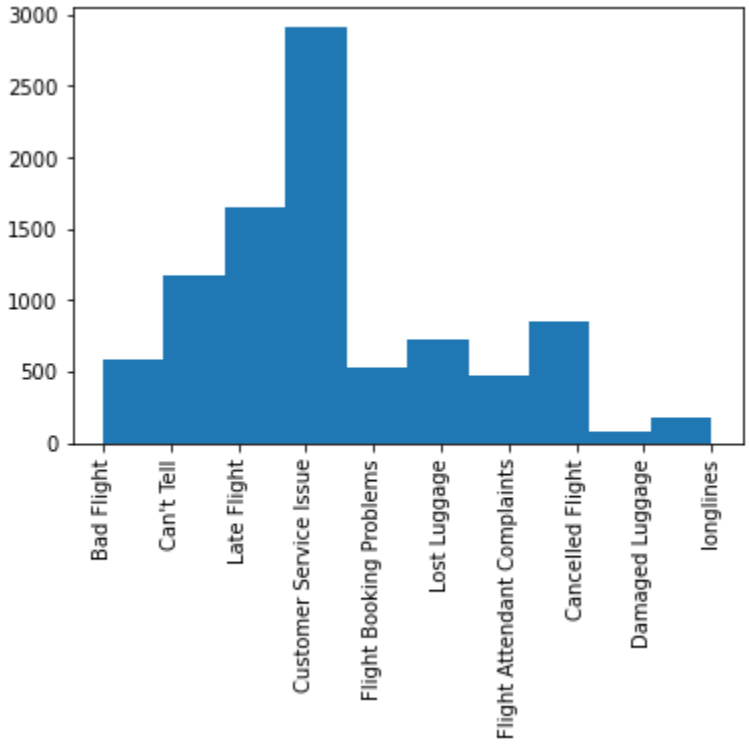
```
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize=(20, 5))

axes[0].hist(ds['airline_sentiment'])
axes[0].set_title("Airline Sentiment")
axes[1].hist(ds['airline_sentiment_confidence'])
axes[1].set_title("Airline Sentiment Confidence")
```





6. Visualizing negative reasons – Customer service, late flights, and cancelled flights seem to have the biggest effect on the customer dissatisfaction.

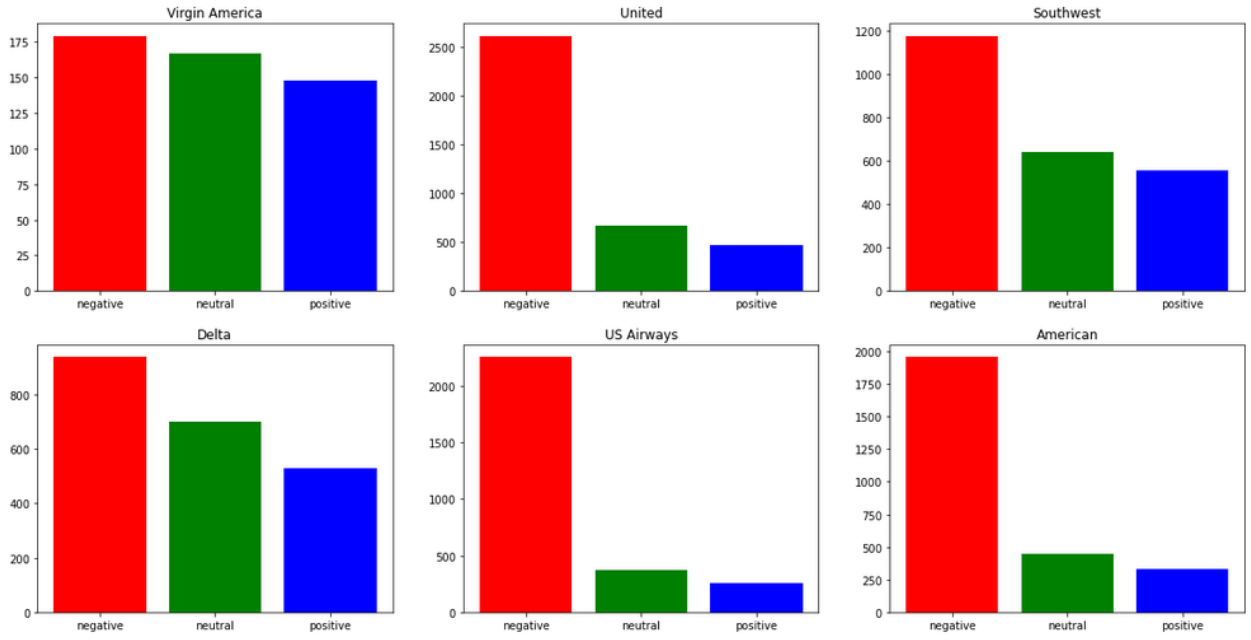
```
plt.hist(ds[ds['negativereason'] != 'NoReason']['negativereason'])
plt.xticks(rotation=90
```



7. Visualizing Tweet sentiment by airline – These histograms might not give you a correct at the first glance. For example, if you compare ‘Virgin America’ with ‘United’ you might think that ‘Virgin America’ is far better but please note the number of total tweets as well. ‘United’ has close to 4000 tweets while ‘Virgin America’ has less than thousand tweets.

```
ds['airline'].unique()
airlines = ['Virgin America', 'United', 'Southwest', 'Delta', 'US Airways', 'American']


fig, axes = plt.subplots(nrows = 2, ncols = 3, figsize=(20, 10))
rows = 0
cols = 0
Index = [1,2,3]
```

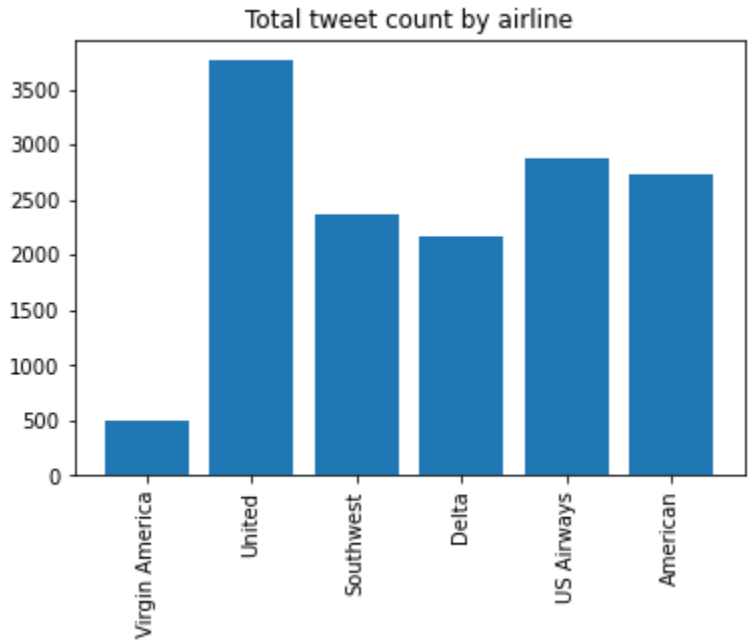
```
for x in airlines:
    count = ds[ds['airline'] == x]['airline_sentiment'].value_counts()
    axes[rows, cols].bar(Index,count, color=['red', 'green', 'blue'])
    axes[rows, cols].set_xticks(Index)
    axes[rows, cols].set_xticklabels(['negative','neutral','positive'])
    axes[rows, cols].set_title(x)

    if cols < 2:
        cols += 1
    else:
        rows += 1
        cols = 0
```



8. Visualizing Tweet count by airline – This is to understand that some airlines naturally have more negative sentiments based on their total number of tweets.

```
tweet_count = []

for x in airlines:
    tweet_count.append(len(ds[ds['airline'] == x]))
```

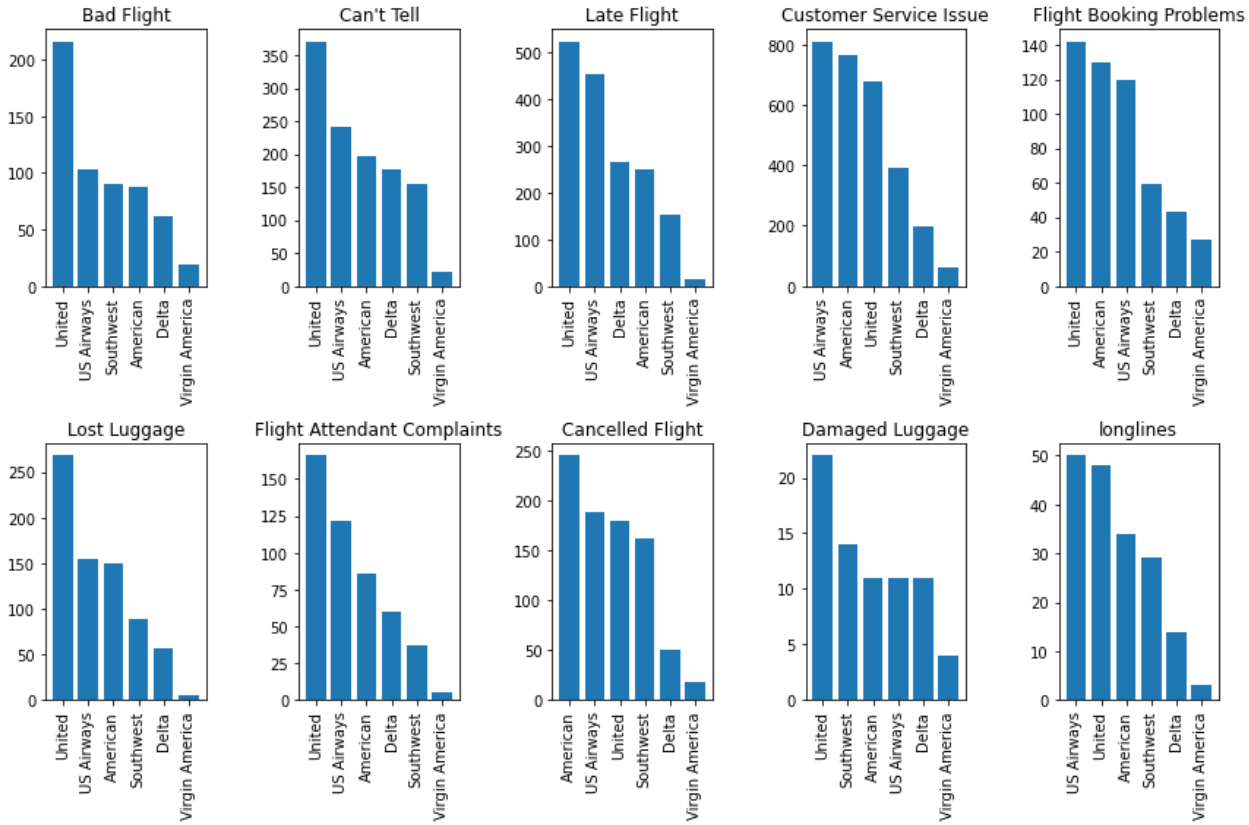
```
tweet_count


plt.bar(airlines,tweet_count)
plt.xticks(rotation=90)
plt.title('Total tweet count by airline')
```



9. Visualizing negative reasons by airline – This is if you are interested in specific negative reasons and want to drill down on how one airline does compared to the others.

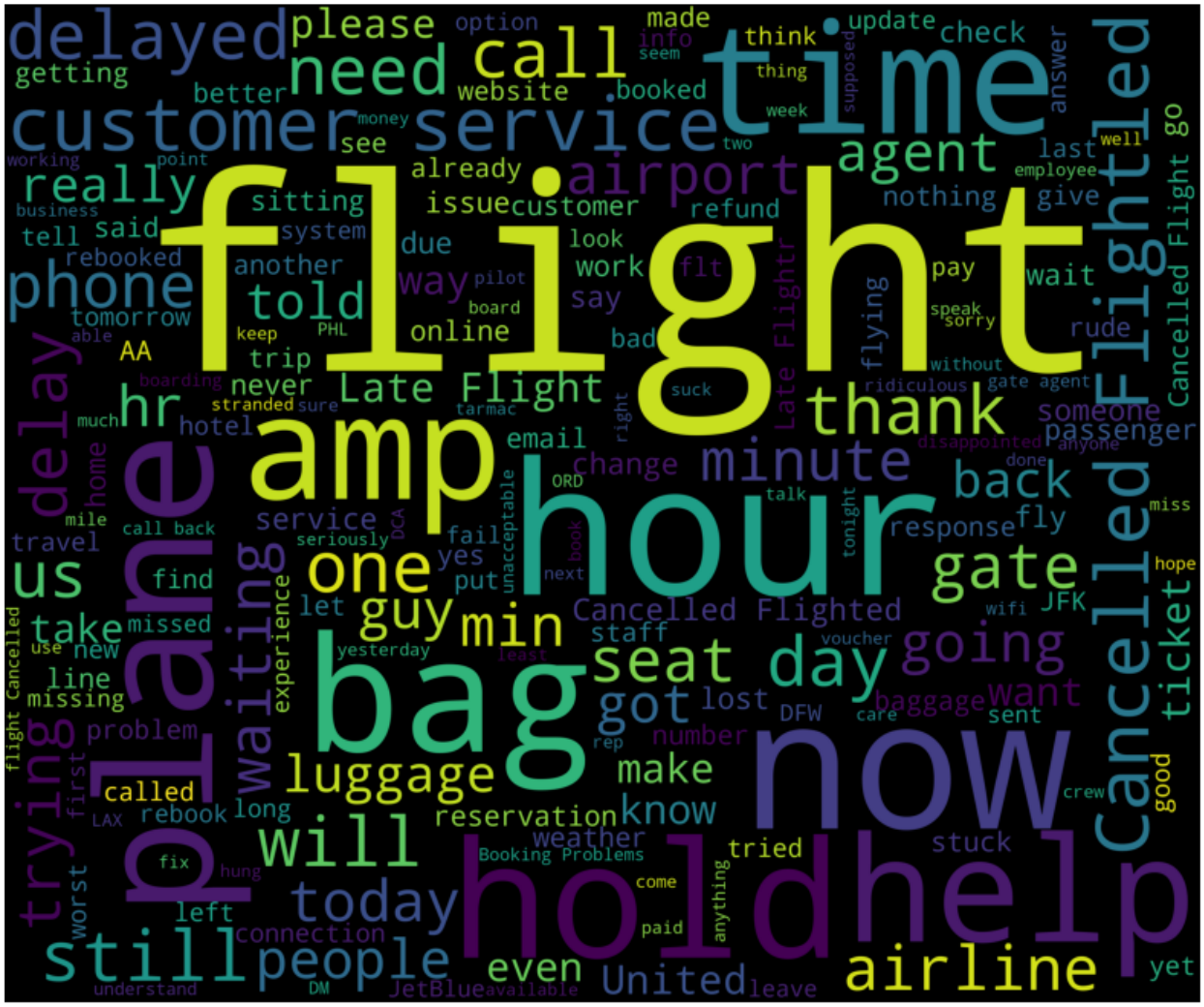
```
ds[ds['negativereason'] != 'NoReason']['negativereason'].unique()
reasons = ['Bad Flight', "Can't Tell", 'Late Flight', 'Customer Service Issue', 'Flight Booking Problems',
      'Lost Luggage', 'Flight Attendant Complaints', 'Cancelled Flight', 'Damaged Luggage', 'longlines']

fig, axes = plt.subplots(nrows = 2, ncols = 5, figsize=(12, 8))
rows = 0
cols = 0
Index = [1,2,3,4,5,6]

for x in reasons:
```

```
    count = ds[ds['negativereason'] == x]['airline'].value_counts()
    axes[rows, cols].bar(Index, count)
    axes[rows, cols].set_xticks(Index)
    plt.setp(axes[rows, cols].xaxis.get_majorticklabels(), rotation=90)
    axes[rows, cols].set_xticklabels(list(count.index))
    axes[rows, cols].set_title(x)

    if cols < 4:
        cols += 1
    else:
        rows += 1
        cols = 0

plt.tight_layout(rect=[0, 0, 1, 1])
```



10. Wordcloud for negative reasons – This is an easy way to spot comments about which specific words contribute most to the overall negative sentiment. It clearly shows how words like ‘flights’, ‘hour’, ‘bag’, ‘time’, and ‘customer service’ are more prominent than the others.

```
def make_wordcloud(sentiment):
    words = ' '.join(ds[ds['airline_sentiment']==sentiment]['text'])

    cleaned_word = " ".join([word for word in words.split()
                             if 'http' not in word
                                and not word.startswith('@')
                             ])

    wordcloud = WordCloud(stopwords=STOPWORDS,
                          background_color='black',
                          width=3000,
                          height=2500
                         ).generate(cleaned_word)

    plt.figure(1,figsize=(15, 12))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()

make_wordcloud('negative')
```



11. Modeling – First, I started with some classifiers to predict the sentiment. I used three different classifiers in my analysis. Using the Decision Tree classifier, I managed to achieve 68% accuracy. Also, you can see that negative sentiments have a 80% accuracy while positive and neutral ones have much less accuracy.

```
from sklearn.feature_extraction.text import CountVectorizer


def cleaned_tweets(tweet):
    words = re.sub("[^a-zA-Z]", " ",tweet).lower().split()
    sentence = [x for x in words if not x in STOPWORDS]
    return( " ".join(sentence))
```

```
# Creating new column with clean text
ds['clean_text']=ds['text'].apply(lambda x: cleaned_tweets(x))

train,test = train_test_split(ds,test_size=0.2,random_state=5)

CV = CountVectorizer(analyzer = "word")
train_features = CV.fit_transform(list(train['clean_text']))
test_features = CV.transform(list(test['clean_text']))

classifier = DecisionTreeClassifier()
Accuracy=[]
Model=[]

def cleaned_tweets(classifier):
    fit = classifier.fit(train_features,train['airline_sentiment'])
    pred = fit.predict(test_features)
    accuracy = accuracy_score(pred,test['airline_sentiment'])
    Accuracy.append(accuracy)
    Model.append(classifier.__class__.__name__)
    print('Accuracy of '+classifier.__class__.__name__+'is '+str(accuracy))
    print(classification_report(pred,test['airline_sentiment']))
    cm=confusion_matrix(pred , test['airline_sentiment'])
    plt.figure()
    plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True,cmap=plt.cm.Reds)
    plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')
    plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)
    plt.show()

cleaned_tweets(classifier)
```

```
Accuracy of DecisionTreeClassifieris 0.6876084692814994
              precision    recall  f1-score   support

    negative       0.80      0.80      0.80      1837
     neutral       0.44      0.45      0.45       590
    positive       0.56      0.56      0.56       454

    accuracy                           0.69      2881
   macro avg       0.60      0.60      0.60      2881
weighted avg       0.69      0.69      0.69      2881
```

12. I got a better accuracy of 76% using the Random Forest classifier. This time, the accuracy of negative sentiments is 91%.

```
classifier = RandomForestClassifier(n_estimators=200)
Accuracy=[]
Model=[]


cleaned_tweets(classifier)
```

```
Accuracy of RandomForestClassifieris 0.7660534536619229
              precision    recall  f1-score   support

    negative       0.91      0.81      0.86      2062
     neutral       0.44      0.62      0.52       432
    positive       0.60      0.70      0.65       387

    accuracy                           0.77      2881
   macro avg       0.65      0.71      0.67      2881
weighted avg       0.80      0.77      0.78      2881
```

13. I got a somewhat similar accuracy of 74% using the Linear SVM classifier.

```
from scipy.sparse import hstack


cv = CountVectorizer(ngram_range=(1,2))
vectorized_data = cv.fit_transform(ds['clean_text'])
```

```
indexed_data = hstack((np.array(range(0,vectorized_data.shape[0]))[:,None],
vectorized_data))

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = ds['airline_sentiment'].apply(sentiment2target)

data_train, data_test, targets_train, targets_test = train_test_split(indexed_data, targets,
test_size=0.4,
                                                      random_state=0)
data_train_index = data_train[:,0]
data_train = data_train[:,1:]
data_test_index = data_test[:,0]
data_test = data_test[:,1:]

from sklearn import svm
from sklearn.multiclass import OneVsRestClassifier

clf = OneVsRestClassifier(svm.SVC(gamma=0.01, C=100., probability=True,
class_weight='balanced', kernel='linear'))
clf_output = clf.fit(data_train, targets_train)

# Model evalutation
clf.score(data_test, targets_test)
```

```
0.7424505380076363
```

14. Next, I used Deep Learning models. First, I tried a sequential model with embeddings. As you can see, I couldn't produce good results with this. Probably if I clean up the tweets better and do more fine tuning it will produce better results.

```
train,test = train_test_split(ds,test_size=0.2,random_state=5)

from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

train_texts = list(train['clean_text'])

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = train['airline_sentiment'].apply(sentiment2target)
train_labels = list(targets)

maxlen = 100
training_samples = 9500
validation_samples = 2000
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_texts)
sequences = tokenizer.texts_to_sequences(train_texts)
```

```
word_index = tokenizer.word_index
data = pad_sequences(sequences, maxlen=maxlen)

train_labels = np.asarray(train_labels)

indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
train_labels = train_labels[indices]

x_train = data[:training_samples]
y_train = train_labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = train_labels[training_samples: training_samples + validation_samples]

embedding_dim = 100

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
```

```
                validation_data=(x_val, y_val))

# Evaluating the model
test_texts = list(test['clean_text'])

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = test['airline_sentiment'].apply(sentiment2target)
test_labels = list(targets)

sequences = tokenizer.texts_to_sequences(test_texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(test_labels)

model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 0s 2ms/step - loss: -343.6100 - acc: 0.5988
[-343.6099548339844, 0.5987504124641418]
```

15. Following that, I tried another deep learning model with a LSTM layer. This time the accuracy increased to 66% but this is still less accurate than the classifiers.

```
from keras.layers import LSTM

max_features = 10000
model = Sequential()
model.add(Embedding(max_features, 32))
```

```
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))

model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 2s 18ms/step - loss: -1.6842 - acc: 0.6668
[-1.6841572523117065, 0.6667823791503906]
```

16. Finally, I tried a deep learning model with a simple 1D convnet. This model actually had worse results than the previous two. Overall, Random forest classifier and Linear SVM classifier got the best results with this dataset. Perhaps, deep learning models will perform better with a much larger dataset.

```
from keras import layers
from keras.optimizers import RMSprop

max_len = 100

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
```

```
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(lr=1e-4), loss='binary_crossentropy',
metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))

model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 0s 3ms/step - loss: 0.3702 - acc: 0.5748
[0.370205283164978, 0.574800431728363]
```

Conclusion

In the recent past we have seen a lot of negative sentiment towards the US airline service due to some controversial incidents. Not just in US, but overall, in the entire world, most people are dissatisfied with the airline service. My goal in this analysis was to build a model that can accurately predict the overall sentiment of a user tweet based on the raw text. I used both classifiers and deep learning models to analyze and I got the best results using a Random forest classifier and a Linear SVM classifier on their own. Both the models had better accuracy levels with negative sentiments as a vast majority of comments were negative. We can probably improve the accuracy with a much larger dataset, so we have enough positive and neutral records to improve on. Further, we can do more cleaning on the raw text, so we extract only the most meaningful words. One should be able to use these models in a real-world environment as well.

Q & A

- Where is this data taken from ?
  Dataset was obtained from Kaggle. However, this has originally been published on Crowdflower.

- Did you have enough resources to do all the required computations ?

This dataset only had 14k records. I could perform all the text analysis functions on the full dataset using my Dell server with 32GB RAM. I think this dataset can be parsed with much less resources too.

- How did you select features ?
  Feature selection wasn't that hard as the number of variables were small. Some features had a lot of null values to be of any use. My analysis was completely based on the 'text' feature which included the raw text from the user's tweet.

- Why did you choose those specific classifiers for your models?
  Random forest and Decision tree classifiers are well known and commonly used for most classification tasks. I read some work related to Linear SVM and decided to try that as well.

- Which model produced better results ?
  In my case it was the Random forest classifier with 74% accuracy. However, as we continue to improve models our, usually deep learning models outperform other methods.

- What are the biggest reasons you have identified for negative sentiments?
  A vast majority of negative sentiments were due to bad customer service. Late flights and cancelled flights also had contributed a lot towards negative sentiments. Looking at the wordcloud you can find more reasons like luggage related issues.

- Are your models better at identifying a specific type of sentiments?
  When I look at the results it seems that these models have performed better with negative sentiments. I am not sure if this is simply due to the availability of more data with negative sentiments or if it's simply a coincidence.

- Did you find it easier to use Python for this analysis ?
  Of course. There are so many different python libraries available for various tasks related to analytics. For example, there is Pandas for analytics, Numpy for numeric operations, Matplotlib and SNS for plotting, SciKitLearn to build and test models etc.

- How good are your system's recommendations ?

It could only achieve accuracy levels as good as 74%. However, all the models I tried during my analysis could be improved with more fine tuning and in-depth training. For example, I only used 10 epochs for my deep learning models.

How can you improve these models ?
One way to improve would be to simply use more data from Twitter directly using their API. Another would be to clean data better so we can get rid of more meaningless words apart from the standard stopwords.

References

- https://www.geeksforgeeks.org/twitter-sentiment-analysis-using-python/

  This article covers the sentiment analysis of any topic by parsing the tweets fetched from Twitter using Python.

- https://towardsdatascience.com/twitter-sentiment-analysis-classification-using-nltk-python-fa912578614c

  This is a Twitter sentiment analysis attempt using natural language processing techniques.

- https://www.pluralsight.com/guides/building-a-twitter-sentiment-analysis-in-python

  Another Twitter sentiment analysis using Python based machine learning libraries.

- https://medium.com/@francesca_lim/twitter-u-s-airline-sentiment-analysis-using-keras-and-rnns-1956f42294ef

  This is a Twitter U.S. Airline Sentiment Analysis using Keras and RNNs.

- https://www.datasciencecentral.com/profiles/blogs/sentiment-analysis-of-airline-tweets

  This is a detailed guide on sentiment analysis for airline tweets.

- https://ipullrank.com/step-step-twitter-sentiment-analysis-visualizing-united-airlines-pr-crisis/

  This post goes into great lengths to explain not only the sentiment analysis process but also how to create an application to collect user sentiments.

- https://www.kaggle.com/parthsharma5795/comprehensive-twitter-airline-sentiment-analysis

  This is a notebook analyzing the same dataset found on Kaggle. He uses a classification approach for his analysis.

- https://www.kaggle.com/anjanatiha/sentiment-analysis-with-lstm-cnn

  This notebook tries to perform sentiment analysis using LSTM & CNN.

- https://www.kaggle.com/mrisdal/exploring-audience-text-length

  Another effort to use audience & tweet length to perform sentiment analysis.

- https://www.kaggle.com/langkilde/linear-svm-classification-of-sentiment-in-tweets

  This notebook uses Linear SVM classification for sentiment analysis.

Appendix I – Dataset

Dataset - https://www.kaggle.com/crowdflower/twitter-airline-sentiment

There is no codebook. Below are the variables.

- tweet_id
- airline_sentiment
- airline_sentiment_confidence
- negativereason
- negativereason_confidence
- airline
- airline_sentiment_gold
- name
- negativereason_gold
- retweet_count
- text
- tweet_coord
- tweet_created
- tweet_location
- user_timezone

Appendix II – Complete Code

```
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
import numpy as np
from wordcloud import WordCloud,STOPWORDS
import re
import nltk
from nltk.corpus import stopwords
from sklearn.model_selection import train_test_split
from mlxtend.plotting import plot_confusion_matrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
```

Loading Data

```
data = pd.read_csv('Tweets.csv')
data.head()
```

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	name
0	570306000000000000	neutral	1.0000	NaN	NaN	Virgin America	NaN	cairdin
1	570301000000000000	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnardino
2	570301000000000000	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonnalynn
3	570301000000000000	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnardino
4	570301000000000000	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnardino

Data Cleanup

Checking for null values

```
data.isnull().sum()
```

```
tweet_id                          0
airline_sentiment                 0
airline_sentiment_confidence      0
negativereason                 5462
negativereason_confidence      4118
airline                           0
airline_sentiment_gold        14600
name                              0
negativereason_gold           14608
retweet_count                     0
text                              0
tweet_coord                   13621
tweet_created                     0
tweet_location                 4733
user_timezone                  4820
dtype: int64
```

```
len(data)
```

```
14640
```

```
#### 'airline_sentiment_gold', 'negativereason_gold', and 'tweet_coord' seem to have too many null values
data.drop(['airline_sentiment_gold', 'negativereason_gold', 'tweet_coord'], axis=1, inplace=True)
list(data.columns)
```

```
['tweet_id',
 'airline_sentiment',
 'airline_sentiment_confidence',
 'negativereason',
 'negativereason_confidence',
 'airline',
 'name',
 'retweet_count',
 'text',
 'tweet_created',
 'tweet_location',
 'user_timezone']
```

```
# Replacing NaNs in 'negativereason_confidence' with 0
data['negativereason_confidence'] = data['negativereason_confidence'].fillna(0)
data['negativereason_confidence'].isnull().sum()
```

```
0
```

```
len(data[data['airline_sentiment_confidence'] > 0.5])
```

```
14404
```

```
len(data[data['negativereason_confidence'] > 0.5])
```

```
7397
```

Based on the output above we can see that half the records don't have a good negative reason confidence. Hence, we won't rely much on that feature. However, Airline Sentiment Confidence on the other hand, has a lot of records with more than 0.5 rating which we will use for further analysis.

```
ds = data[data['airline_sentiment_confidence'] > 0.5]
len(ds)
```

```
14404
```

```
# Replacing remaining null values with arbitrary values

ds['negativereason'] = ds['negativereason'].fillna('NoReason')
ds['tweet_location'] = ds['tweet_location'].fillna('NoLocation')
ds['user_timezone'] = ds['user_timezone'].fillna('NoTimezone')
```

```
ds.isnull().values.any()
```

```
False
```

```
len(ds[ds['retweet_count'] != 0])
```

```
758
```

```
# Removing 'retweet_count' column as well as it is mostly zeroes.
ds.drop(['retweet_count'], axis=1, inplace=True)
```

Data Exploration

Checking the number of unique values in each column

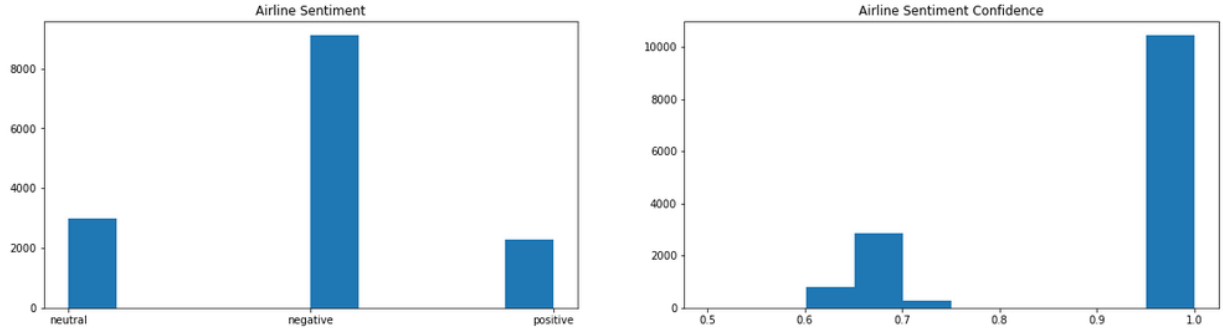
```
for x in list(ds.columns):
    print(x, ":", len(ds[x].unique()))
```

```
tweet_id : 2424
airline_sentiment : 3
airline_sentiment_confidence : 868
negativereason : 11
negativereason_confidence : 1405
airline : 6
name : 7624
text : 14197
tweet_created : 6750
tweet_location : 3054
user_timezone : 85
```

Visualizing airline sentiments

```
fig, axes = plt.subplots(nrows = 1, ncols = 2, figsize=(20, 5))

axes[0].hist(ds['airline_sentiment'])
axes[0].set_title("Airline Sentiment")
axes[1].hist(ds['airline_sentiment_confidence'])
axes[1].set_title("Airline Sentiment Confidence")
```

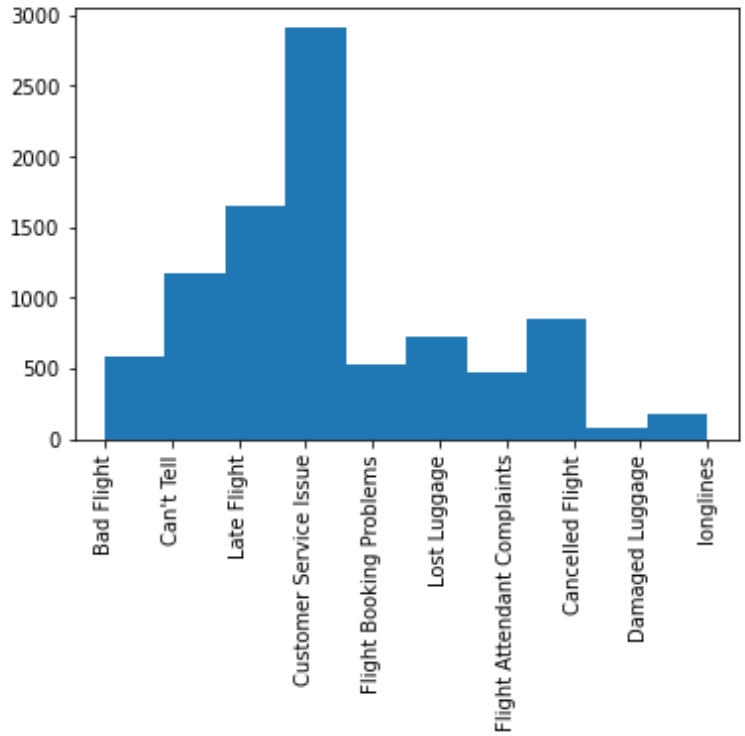
```
Text(0.5, 1.0, 'Airline Sentiment Confidence')
```



Visualizing Negative Reasons

```
plt.hist(ds[ds['negativereason'] != 'NoReason']['negativereason'])
plt.xticks(rotation=90)
```

```
([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], <a list of 10 Text major ticklabel objects>)
```

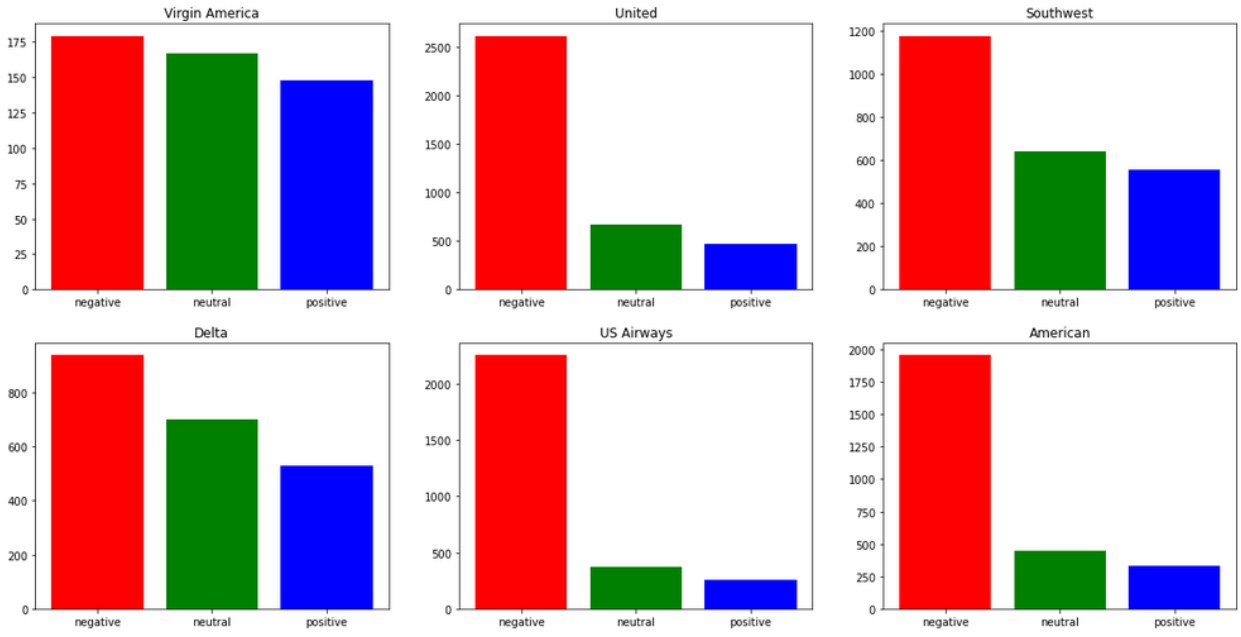


Visualizing Tweet sentiment by Airline

```
ds['airline'].unique()
```

```
array(['Virgin America', 'United', 'Southwest', 'Delta', 'US Airways',
       'American'], dtype=object)
```

```
airlines = ['Virgin America', 'United', 'Southwest', 'Delta', 'US Airways', 'American']

fig, axes = plt.subplots(nrows = 2, ncols = 3, figsize=(20, 10))
rows = 0
cols = 0
Index = [1,2,3]

for x in airlines:
    count = ds[ds['airline'] == x]['airline_sentiment'].value_counts()
    axes[rows, cols].bar(Index,count, color=['red', 'green', 'blue'])
    axes[rows, cols].set_xticks(Index)
    axes[rows, cols].set_xticklabels(['negative','neutral','positive'])
    axes[rows, cols].set_title(x)

    if cols < 2:
        cols += 1
    else:
        rows += 1
        cols = 0
```



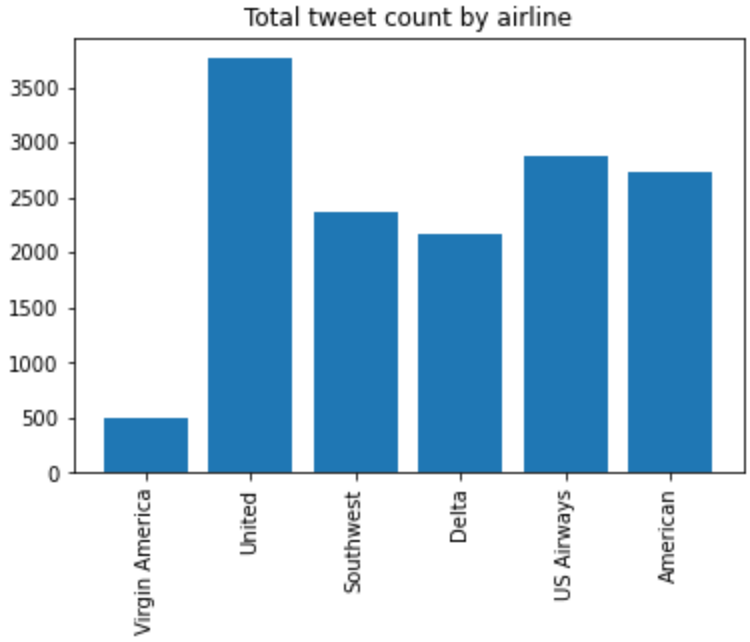
Visualizing Tweet count by Airline

```
tweet_count = []

for x in airlines:
    tweet_count.append(len(ds[ds['airline'] == x]))

tweet_count
```

```
[494, 3760, 2372, 2169, 2876, 2733]
```

```
plt.bar(airlines,tweet_count)
plt.xticks(rotation=90)
plt.title('Total tweet count by airline')
```

```
Text(0.5, 1.0, 'Total tweet count by airline')
```

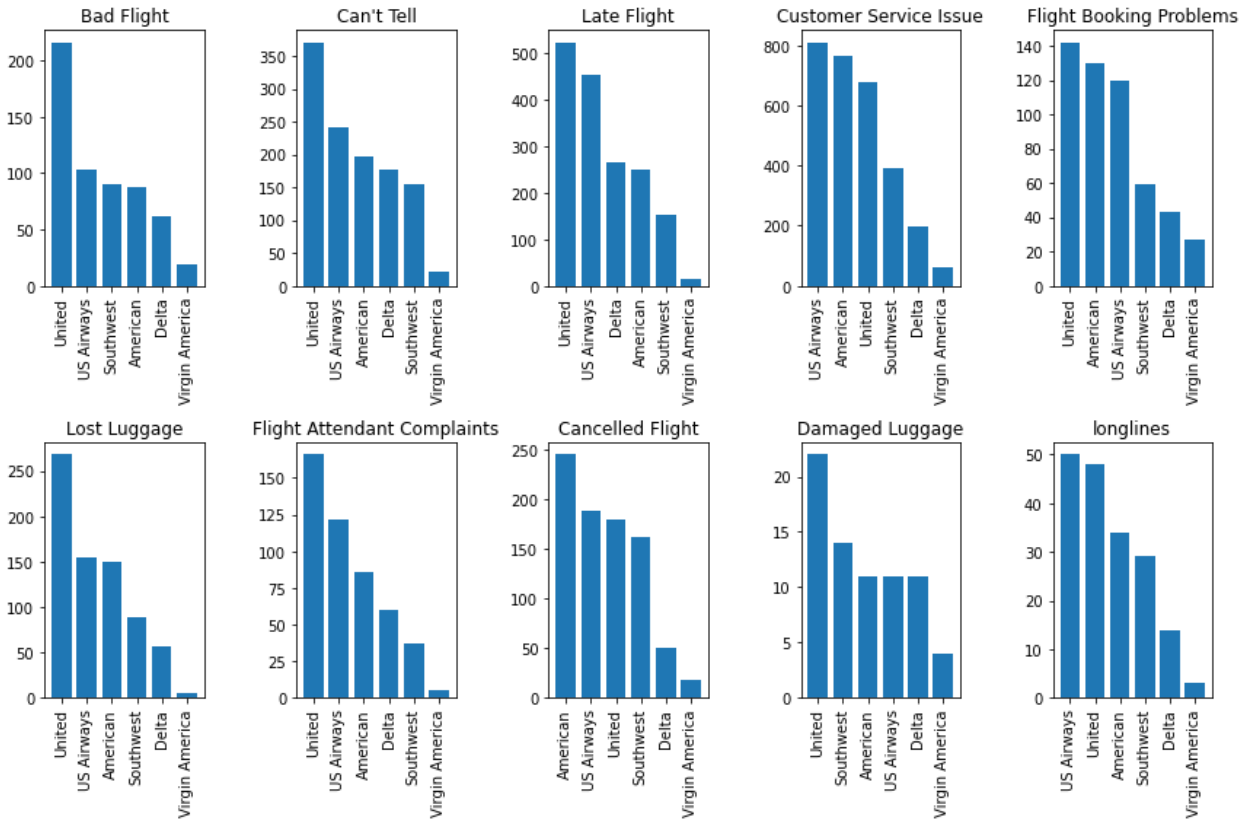


Visualizing Negative Reasons by Airline

```
ds[ds['negativereason'] != 'NoReason']['negativereason'].unique()
```

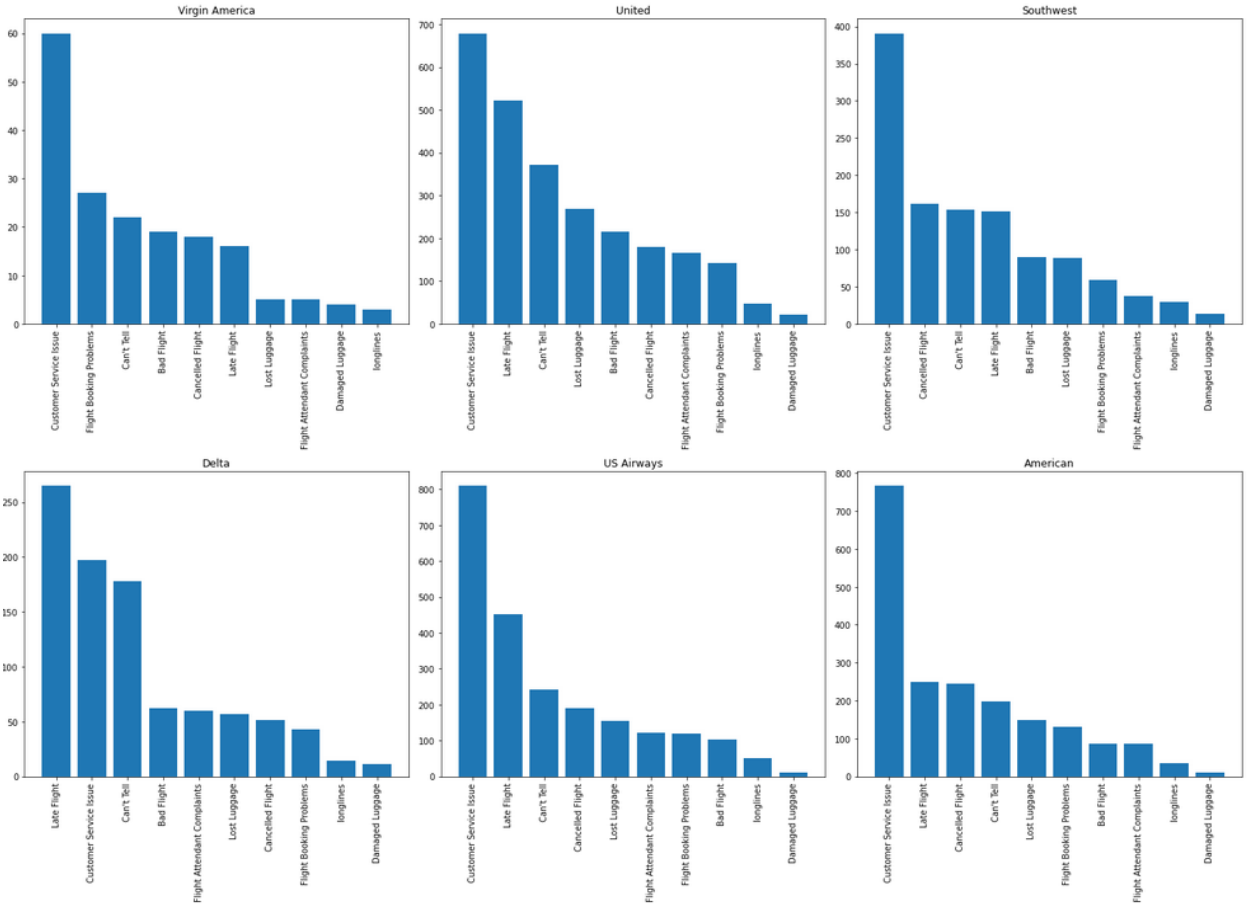
```
array(['Bad Flight', "Can't Tell", 'Late Flight',
       'Customer Service Issue', 'Flight Booking Problems',
       'Lost Luggage', 'Flight Attendant Complaints', 'Cancelled Flight',
       'Damaged Luggage', 'longlines'], dtype=object)
```

```
reasons = ['Bad Flight', "Can't Tell", 'Late Flight', 'Customer Service Issue', 'Flight Booking Problems',
       'Lost Luggage', 'Flight Attendant Complaints', 'Cancelled Flight', 'Damaged Luggage', 'longlines']

fig, axes = plt.subplots(nrows = 2, ncols = 5, figsize=(12, 8))
rows = 0
cols = 0
Index = [1,2,3,4,5,6]

for x in reasons:
    count = ds[ds['negativereason'] == x]['airline'].value_counts()
    axes[rows, cols].bar(Index, count)
    axes[rows, cols].set_xticks(Index)
    plt.setp(axes[rows, cols].xaxis.get_majorticklabels(), rotation=90)
    axes[rows, cols].set_xticklabels(list(count.index))
    axes[rows, cols].set_title(x)

    if cols < 4:
        cols += 1
    else:
        rows += 1
        cols = 0

plt.tight_layout(rect=[0, 0, 1, 1])
```



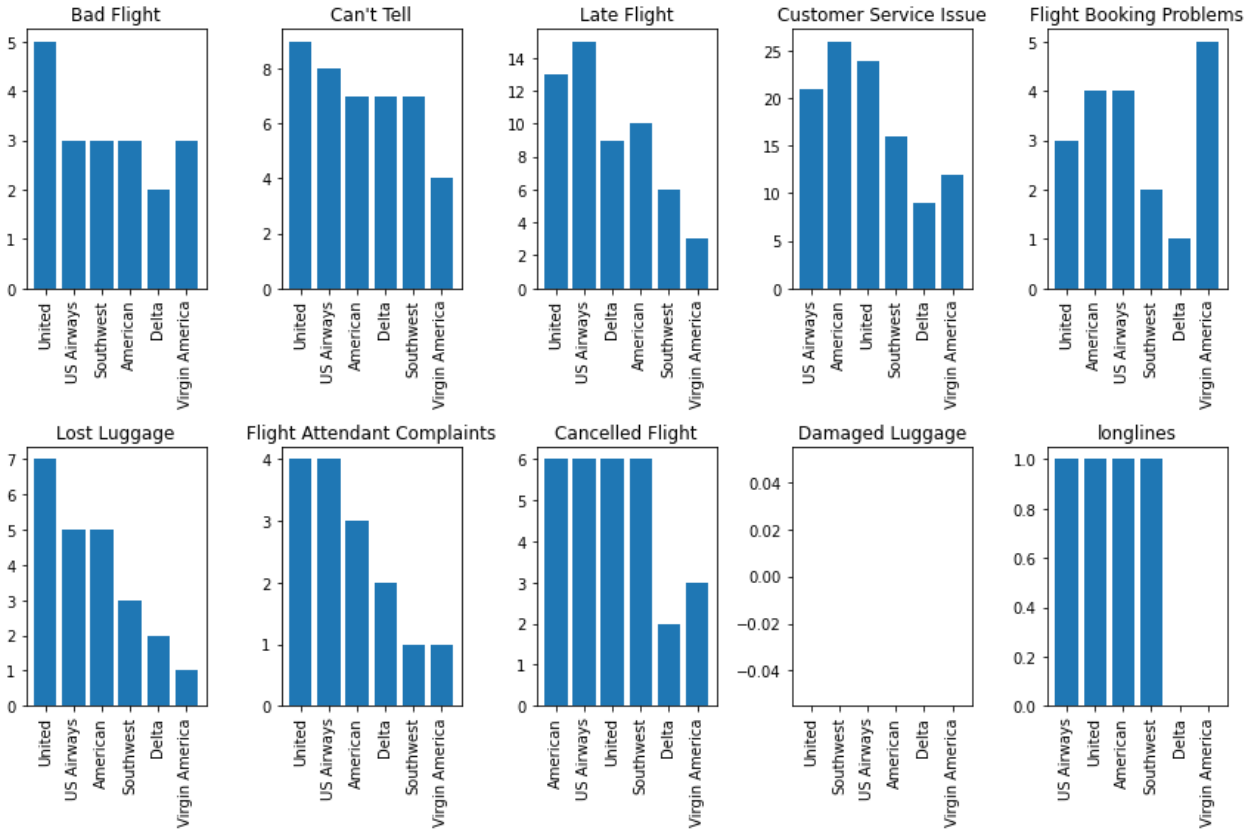
Visualizing Airlines by Negative Reasons

```
fig, axes = plt.subplots(nrows = 2, ncols = 3, figsize=(20, 15))
rows = 0
cols = 0
Index = [1,2,3,4,5,6,7,8,9,10]

for x in airlines:
    count = ds[ds['airline'] == x]['negativereason'].value_counts()
    for n in range(11):
        if count.index[n] == 'NoReason':
            count.drop(count.index[n], inplace=True)
            break
    axes[rows, cols].bar(Index, count)
    axes[rows, cols].set_xticks(Index)
    plt.setp(axes[rows, cols].xaxis.get_majorticklabels(), rotation=90)
    axes[rows, cols].set_xticklabels(list(count.index))
    axes[rows, cols].set_title(x)

    if cols < 2:
        cols += 1
    else:
        rows += 1
        cols = 0

plt.tight_layout(rect=[0, 0, 1, 1])
```



Visualizing Negative Reasons by Airline as a percentage of total tweets

```
import math
total = ds['airline'].value_counts()

fig, axes = plt.subplots(nrows = 2, ncols = 5, figsize=(12, 8))
rows = 0
cols = 0

for x in reasons:
    count = ds[ds['negativereason'] == x]['airline'].value_counts()
    for n in range(6):
        count[n] = math.floor((count[n]/total[n])*100)
    axes[rows, cols].bar(Index, count)
    axes[rows, cols].set_xticks(Index)
    plt.setp(axes[rows, cols].xaxis.get_majorticklabels(), rotation=90)
    axes[rows, cols].set_xticklabels(list(count.index))
    axes[rows, cols].set_title(x)

    if cols < 4:
        cols += 1
    else:
        rows += 1
        cols = 0

plt.tight_layout(rect=[0, 0, 1, 1])
```

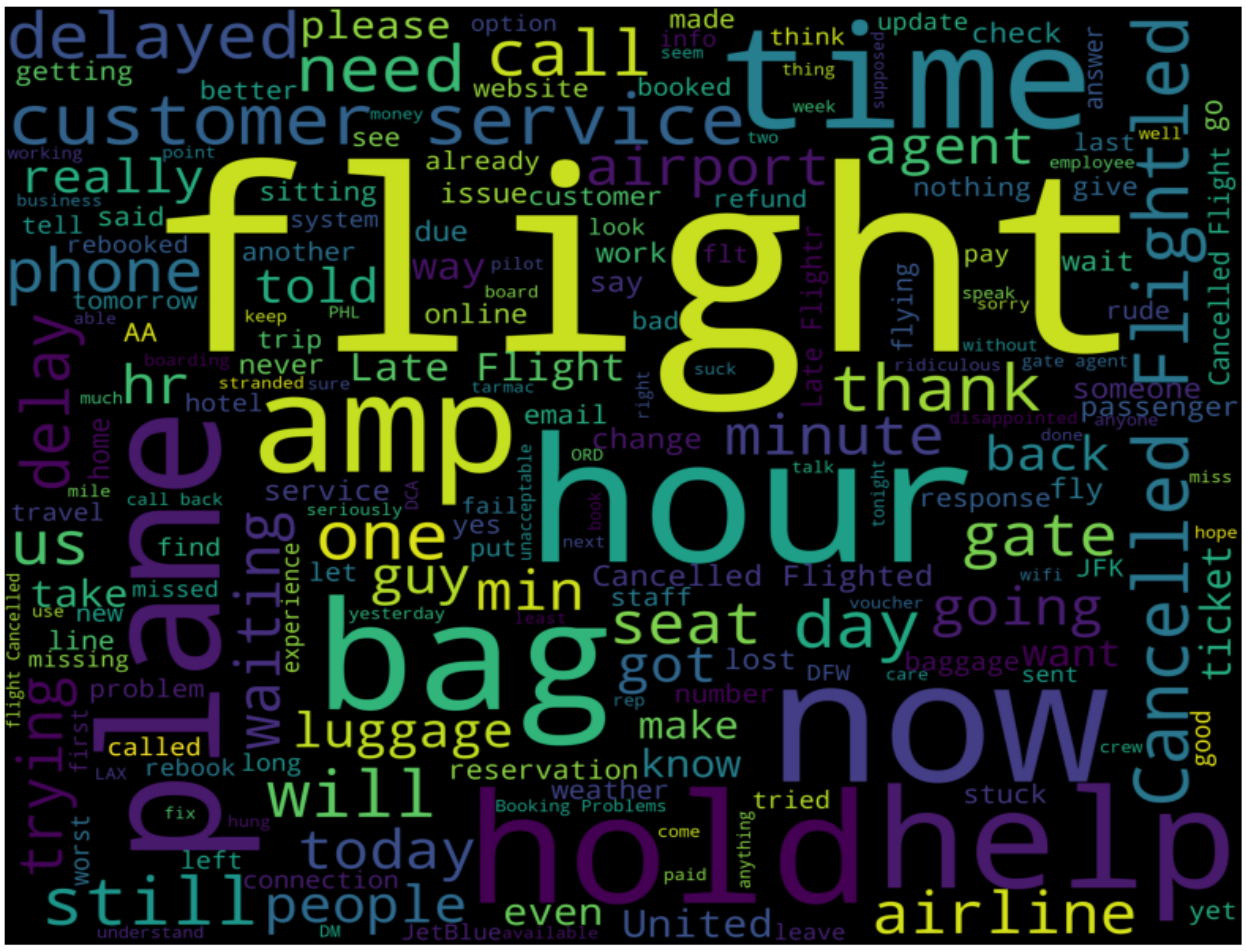


Wordclouds

Wordcloud for negative reasons

```
def make_wordcloud(sentiment):
    words = ' '.join(ds[ds['airline_sentiment']==sentiment]['text'])

    cleaned_word = " ".join([word for word in words.split()
                                if 'http' not in word
                                    and not word.startswith('@')
                                ])

    wordcloud = WordCloud(stopwords=STOPWORDS,
                          background_color='black',
                          width=3000,
                          height=2500
                         ).generate(cleaned_word)

    plt.figure(1,figsize=(15, 12))
    plt.imshow(wordcloud)
    plt.axis('off')
    plt.show()
```

```
make_wordcloud('negative')
```



Modeling

Using classifiers to analyse sentiment based on Tweet text

Using Decision Tree Classifier

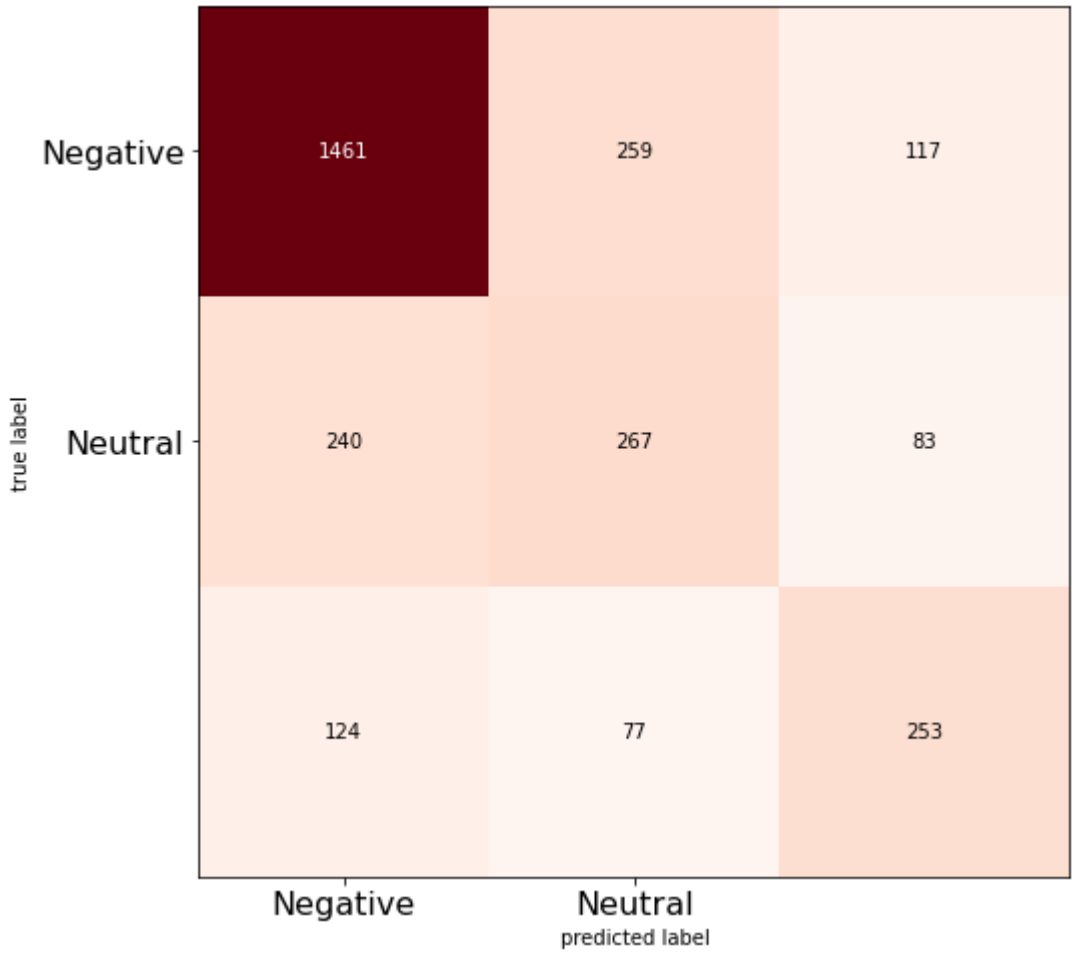
```
from sklearn.feature_extraction.text import CountVectorizer

def cleaned_tweets(tweet):
    words = re.sub("[^a-zA-Z]", " ",tweet).lower().split()
    sentence = [x for x in words if not x in STOPWORDS]
    return( " ".join(sentence))

# Creating new column with clean text
ds['clean_text']=ds['text'].apply(lambda x: cleaned_tweets(x))
```

```
train,test = train_test_split(ds,test_size=0.2,random_state=5)

CV = CountVectorizer(analyzer = "word")
train_features = CV.fit_transform(list(train['clean_text']))
test_features = CV.transform(list(test['clean_text']))
```

```
classifier = DecisionTreeClassifier()
Accuracy=[]
Model=[]

def cleaned_tweets(classifier):
    fit = classifier.fit(train_features,train['airline_sentiment'])
    pred = fit.predict(test_features)
    accuracy = accuracy_score(pred,test['airline_sentiment'])
    Accuracy.append(accuracy)
    Model.append(classifier.__class__.__name__)
    print('Accuracy of '+classifier.__class__.__name__+'is '+str(accuracy))
    print(classification_report(pred,test['airline_sentiment']))
    cm=confusion_matrix(pred , test['airline_sentiment'])
    plt.figure()
    plot_confusion_matrix(cm,figsize=(12,8), hide_ticks=True,cmap=plt.cm.Reds)
    plt.xticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16,color='black')
    plt.yticks(range(2), ['Negative', 'Neutral', 'Positive'], fontsize=16)
    plt.show()
```

```
cleaned_tweets(classifier)
```

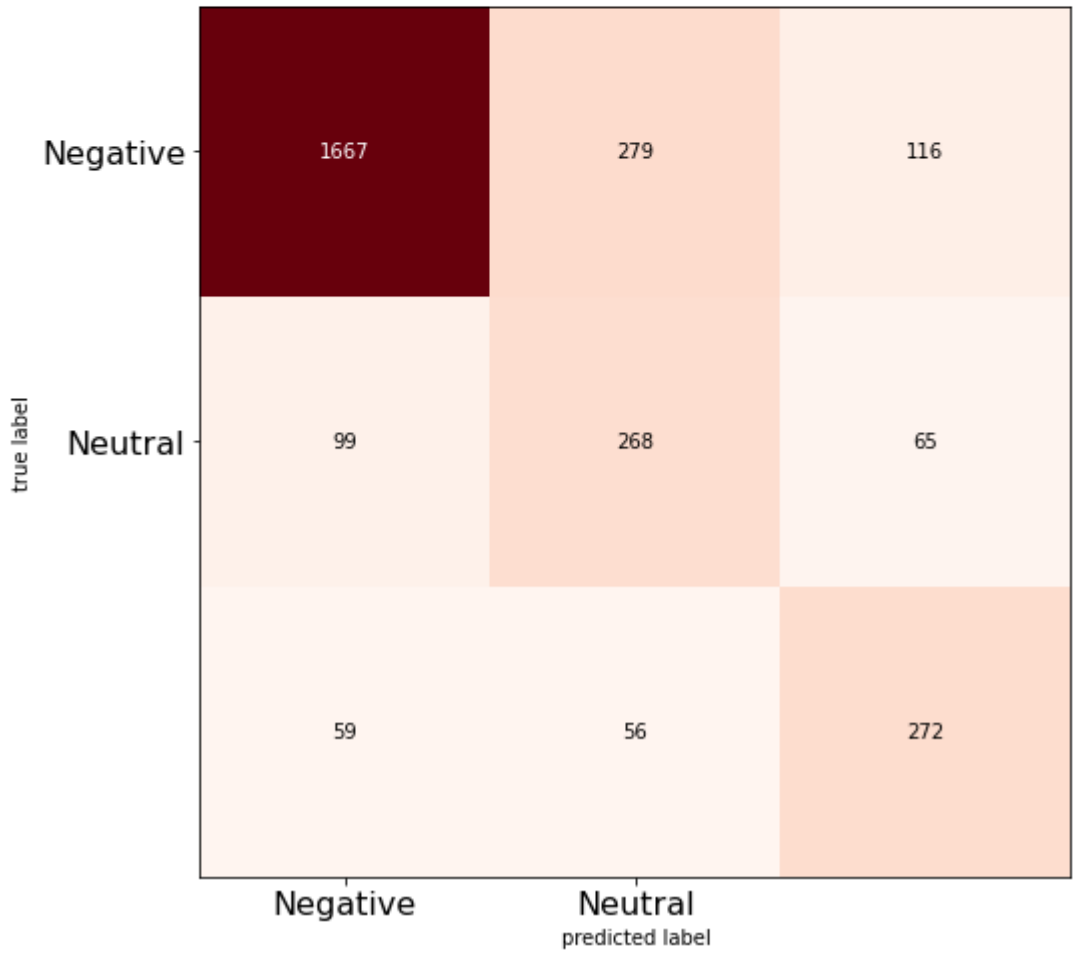
```
Accuracy of DecisionTreeClassifieris 0.6876084692814994
              precision    recall  f1-score   support

    negative       0.80      0.80      0.80      1837
     neutral       0.44      0.45      0.45       590
    positive       0.56      0.56      0.56       454

    accuracy                           0.69      2881
   macro avg       0.60      0.60      0.60      2881
weighted avg       0.69      0.69      0.69      2881
```


Negative
Neutral
true label
1461
259
117
240
267
83
124
77
253
Negative
Neutral
predicted label

Random Forest Classifier

```
classifier = RandomForestClassifier(n_estimators=200)
Accuracy=[]
Model=[]

cleaned_tweets(classifier)
```

```
Accuracy of RandomForestClassifieris 0.7660534536619229
              precision    recall  f1-score   support

    negative       0.91      0.81      0.86      2062
     neutral       0.44      0.62      0.52       432
    positive       0.60      0.70      0.65       387

    accuracy                           0.77      2881
   macro avg       0.65      0.71      0.67      2881
weighted avg       0.80      0.77      0.78      2881


<Figure size 432x288 with 0 Axes>
```



Using Linear SVM classifier

```
from scipy.sparse import hstack

cv = CountVectorizer(ngram_range=(1,2))
vectorized_data = cv.fit_transform(ds['clean_text'])
indexed_data = hstack((np.array(range(0,vectorized_data.shape[0]))[:,None], vectorized_data))

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = ds['airline_sentiment'].apply(sentiment2target)
```

```
data_train, data_test, targets_train, targets_test = train_test_split(indexed_data, targets, test_size=0.4,
                                                                      random_state=0)
data_train_index = data_train[:,0]
data_train = data_train[:,1:]
data_test_index = data_test[:,0]
data_test = data_test[:,1:]
```

```
from sklearn import svm
from sklearn.multiclass import OneVsRestClassifier

clf = OneVsRestClassifier(svm.SVC(gamma=0.01, C=100., probability=True, class_weight='balanced', kernel='linear'))
clf_output = clf.fit(data_train, targets_train)
```

```
# Model evalutation
clf.score(data_test, targets_test)
```

```
0.7424505380076363
```

```
# Verifying with som examples
sentences = cv.transform(list(ds['clean_text'][70:80]))
clf.predict_proba(sentences)
```

```
array([[0.30035218, 0.61802616, 0.08162167],
       [0.29176254, 0.60039504, 0.10784242],
       [0.74972568, 0.13697718, 0.11329713],
       [0.19294116, 0.02584966, 0.78120918],
       [0.23646453, 0.139661  , 0.62387447],
       [0.28257736, 0.58159839, 0.13582425],
       [0.2825697 , 0.58163119, 0.13579911],
       [0.85322885, 0.01785486, 0.12891629],
       [0.28259441, 0.58162851, 0.13577708],
       [0.83015048, 0.14491116, 0.02493836]])
```

Using Deep learning

Training a sequential model with embeddings

```
train,test = train_test_split(ds,test_size=0.2,random_state=5)
print("train -", train.shape, "\n", "test -", test.shape)
```

```
train - (11523, 12)
 test - (2881, 12)
```

```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Embedding, Flatten, Dense

train_texts = list(train['clean_text'])

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = train['airline_sentiment'].apply(sentiment2target)
train_labels = list(targets)

maxlen = 100
training_samples = 9500
validation_samples = 2000
max_words = 10000

tokenizer = Tokenizer(num_words=max_words)
tokenizer.fit_on_texts(train_texts)
sequences = tokenizer.texts_to_sequences(train_texts)

word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 11744 unique tokens.
```

```
data = pad_sequences(sequences, maxlen=maxlen)

train_labels = np.asarray(train_labels)
print('Shape of data tensor:', data.shape)
print('Shape of label tensor:', train_labels.shape)
```

```
Shape of data tensor: (11523, 100)
Shape of label tensor: (11523,)
```

```
indices = np.arange(data.shape[0])
np.random.shuffle(indices)
data = data[indices]
train_labels = train_labels[indices]

x_train = data[:training_samples]
y_train = train_labels[:training_samples]
x_val = data[training_samples: training_samples + validation_samples]
y_val = train_labels[training_samples: training_samples + validation_samples]
```

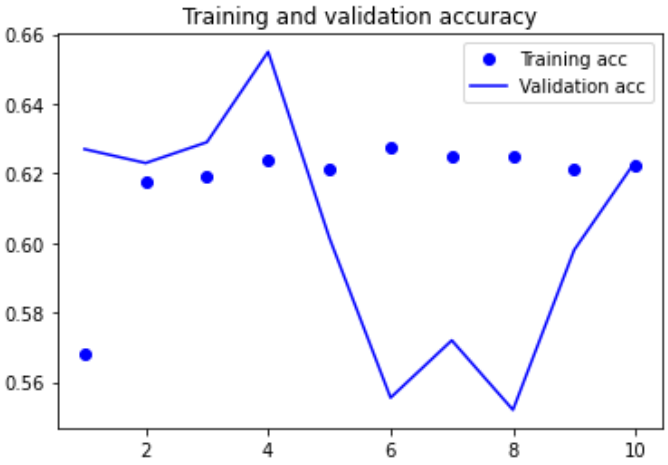
```
embedding_dim = 100

model = Sequential()
model.add(Embedding(max_words, embedding_dim, input_length=maxlen))
model.add(Flatten())
model.add(Dense(32, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

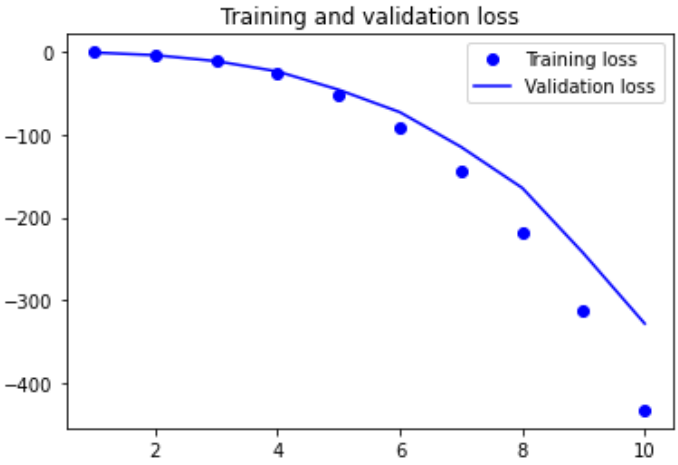
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_4 (Embedding)      (None, 100, 100)          1000000
_________________________________________________________________
flatten_3 (Flatten)          (None, 10000)             0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                320032
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33
=================================================================
Total params: 1,320,065
Trainable params: 1,320,065
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=32,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
297/297 [==============================] - 3s 9ms/step - loss: 0.1035 - acc: 0.5682 - val_loss: -0.5292 - val_acc: 0.6270
Epoch 2/10
297/297 [==============================] - 3s 9ms/step - loss: -2.7027 - acc: 0.6177 - val_loss: -3.7541 - val_acc: 0.6230
Epoch 3/10
297/297 [==============================] - 3s 9ms/step - loss: -10.1847 - acc: 0.6192 - val_loss: -10.9518 - val_acc: 0.629
0
Epoch 4/10
297/297 [==============================] - 3s 9ms/step - loss: -25.5203 - acc: 0.6237 - val_loss: -23.3977 - val_acc: 0.655
0
Epoch 5/10
297/297 [==============================] - 3s 9ms/step - loss: -51.9467 - acc: 0.6211 - val_loss: -45.6026 - val_acc: 0.601
5
Epoch 6/10
297/297 [==============================] - 3s 9ms/step - loss: -90.7190 - acc: 0.6277 - val_loss: -72.7157 - val_acc: 0.555
5
Epoch 7/10
297/297 [==============================] - 3s 9ms/step - loss: -144.6452 - acc: 0.6248 - val_loss: -114.7961 - val_acc: 0.5
720
Epoch 8/10
297/297 [==============================] - 3s 9ms/step - loss: -218.1948 - acc: 0.6251 - val_loss: -164.4126 - val_acc: 0.5
520
Epoch 9/10
297/297 [==============================] - 3s 9ms/step - loss: -312.1018 - acc: 0.6212 - val_loss: -243.5307 - val_acc: 0.5
980
Epoch 10/10
297/297 [==============================] - 3s 9ms/step - loss: -433.2049 - acc: 0.6221 - val_loss: -328.5719 - val_acc: 0.6
235
```

```
import matplotlib.pyplot as plt

def plotting():
    acc = history.history['acc']
    val_acc = history.history['val_acc']
    loss = history.history['loss']
    val_loss = history.history['val_loss']

    epochs = range(1, len(acc) + 1)

    plt.plot(epochs, acc, 'bo', label='Training acc')
    plt.plot(epochs, val_acc, 'b', label='Validation acc')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'bo', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

plotting()
```






```
# Evaluating the model
test_texts = list(test['clean_text'])

def sentiment2target(sentiment):
    return {
        'negative': 0,
        'neutral': 1,
        'positive' : 2
    }[sentiment]

targets = test['airline_sentiment'].apply(sentiment2target)
test_labels = list(targets)

sequences = tokenizer.texts_to_sequences(test_texts)
x_test = pad_sequences(sequences, maxlen=maxlen)
y_test = np.asarray(test_labels)

model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 0s 2ms/step - loss: -343.6100 - acc: 0.5988

[-343.6099548339844, 0.5987504124641418]
```

Training a model with a LSTM layer

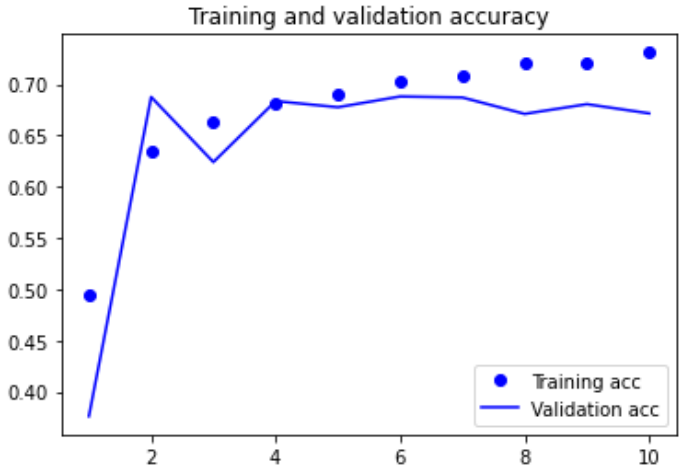
```
from keras.layers import LSTM

max_features = 10000

model = Sequential()
model.add(Embedding(max_features, 32))
model.add(LSTM(32))
model.add(Dense(1, activation='sigmoid'))

model.summary()
```

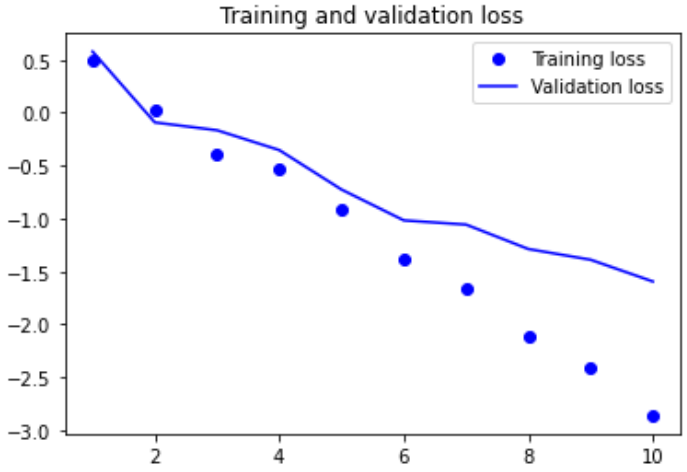
```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
lstm_1 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
75/75 [==============================] - 8s 106ms/step - loss: 0.5025 - acc: 0.4941 - val_loss: 0.5777 - val_acc: 0.3760
Epoch 2/10
75/75 [==============================] - 7s 94ms/step - loss: 0.0182 - acc: 0.6340 - val_loss: -0.0919 - val_acc: 0.6875
Epoch 3/10
75/75 [==============================] - 7s 98ms/step - loss: -0.3858 - acc: 0.6638 - val_loss: -0.1658 - val_acc: 0.6240
Epoch 4/10
75/75 [==============================] - 7s 95ms/step - loss: -0.5290 - acc: 0.6809 - val_loss: -0.3534 - val_acc: 0.6835
Epoch 5/10
75/75 [==============================] - 7s 96ms/step - loss: -0.9159 - acc: 0.6904 - val_loss: -0.7264 - val_acc: 0.6775
Epoch 6/10
75/75 [==============================] - 7s 97ms/step - loss: -1.3804 - acc: 0.7019 - val_loss: -1.0176 - val_acc: 0.6880
Epoch 7/10
75/75 [==============================] - 7s 92ms/step - loss: -1.6657 - acc: 0.7074 - val_loss: -1.0574 - val_acc: 0.6870
Epoch 8/10
75/75 [==============================] - 7s 93ms/step - loss: -2.1114 - acc: 0.7213 - val_loss: -1.2882 - val_acc: 0.6710
Epoch 9/10
75/75 [==============================] - 7s 94ms/step - loss: -2.4025 - acc: 0.7209 - val_loss: -1.3894 - val_acc: 0.6805
Epoch 10/10
75/75 [==============================] - 7s 94ms/step - loss: -2.8612 - acc: 0.7311 - val_loss: -1.5951 - val_acc: 0.6715
```

```
plotting()
```






```
model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 2s 18ms/step - loss: -1.6842 - acc: 0.6668

[-1.6841572523117065, 0.6667823791503906]
```

Training a model with a simple 1D convnet

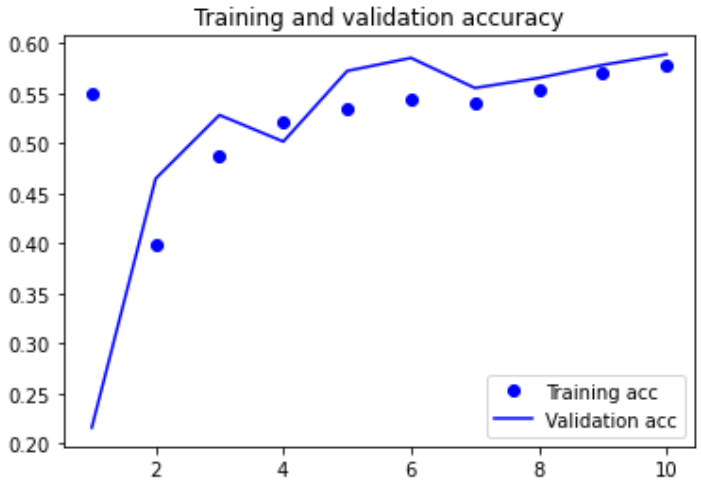
```
from keras import layers
from keras.optimizers import RMSprop

max_len = 100

model = Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))

model.summary()
```

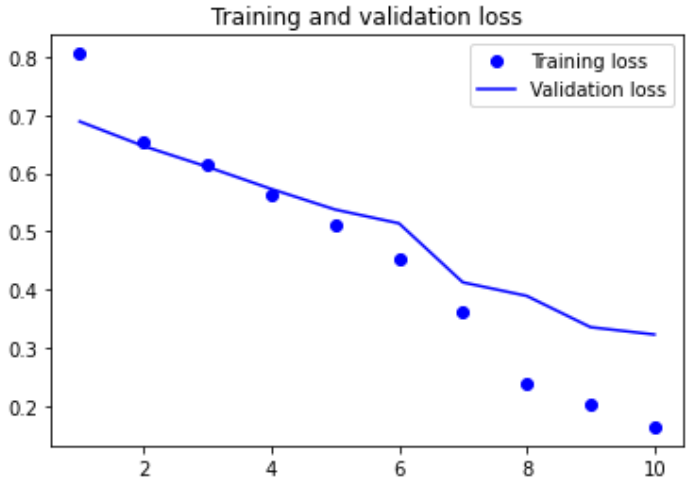
```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_6 (Embedding)      (None, 100, 128)          1280000
_________________________________________________________________
conv1d (Conv1D)              (None, 94, 32)            28704
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, 18, 32)            0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 12, 32)            7200
_________________________________________________________________
global_max_pooling1d (Global (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 1)                 33
=================================================================
Total params: 1,315,937
Trainable params: 1,315,937
Non-trainable params: 0
```

```
model.compile(optimizer=RMSprop(lr=1e-4), loss='binary_crossentropy', metrics=['acc'])

history = model.fit(x_train, y_train,
                    epochs=10,
                    batch_size=128,
                    validation_data=(x_val, y_val))
```

```
Epoch 1/10
75/75 [==============================] - 2s 22ms/step - loss: 0.8053 - acc: 0.5491 - val_loss: 0.6886 - val_acc: 0.2160
Epoch 2/10
75/75 [==============================] - 2s 21ms/step - loss: 0.6547 - acc: 0.3978 - val_loss: 0.6465 - val_acc: 0.4645
Epoch 3/10
75/75 [==============================] - 2s 21ms/step - loss: 0.6139 - acc: 0.4867 - val_loss: 0.6108 - val_acc: 0.5280
Epoch 4/10
75/75 [==============================] - 2s 21ms/step - loss: 0.5636 - acc: 0.5206 - val_loss: 0.5728 - val_acc: 0.5015
Epoch 5/10
75/75 [==============================] - 2s 20ms/step - loss: 0.5094 - acc: 0.5348 - val_loss: 0.5374 - val_acc: 0.5720
Epoch 6/10
75/75 [==============================] - 2s 20ms/step - loss: 0.4522 - acc: 0.5436 - val_loss: 0.5138 - val_acc: 0.5850
Epoch 7/10
75/75 [==============================] - 2s 21ms/step - loss: 0.3623 - acc: 0.5406 - val_loss: 0.4124 - val_acc: 0.5550
Epoch 8/10
75/75 [==============================] - 2s 21ms/step - loss: 0.2384 - acc: 0.5534 - val_loss: 0.3891 - val_acc: 0.5650
Epoch 9/10
75/75 [==============================] - 2s 21ms/step - loss: 0.2037 - acc: 0.5696 - val_loss: 0.3354 - val_acc: 0.5780
Epoch 10/10
75/75 [==============================] - 2s 20ms/step - loss: 0.1642 - acc: 0.5768 - val_loss: 0.3228 - val_acc: 0.5885
```

```
plotting()
```






```
model.evaluate(x_test, y_test)
```

```
91/91 [==============================] - 0s 3ms/step - loss: 0.3702 - acc: 0.5748

[0.370205283164978, 0.574800431728363]
```