ML MAJOR JUNE ML063B12

IMPORTING MODULES:

1) import warnings
warnings.filterwarnings('ignore')

from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = 'all'

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import math

READING FILE:

2) tweet_df = pd.read_csv(r'C:\Users\vamsh\Downloads\Tweets.csv')

3) tweet_df.head()

```
In [5]: tweet_df.head()
```

Out[5]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	na
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	cai
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jnar
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonna
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jnar
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jnar

INVESTIGATE THE DATAFRAME:

```
In [6]:  tweet_df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14640 entries, 0 to 14639
Data columns (total 15 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   airline_sentiment_gold        40 non-null     object 
 7   name                          14640 non-null  object 
 8   negativereason_gold           32 non-null     object 
 9   retweet_count                 14640 non-null  int64  
 10  text                          14640 non-null  object 
 11  tweet_coord                   1019 non-null   object 
 12  tweet_created                 14640 non-null  object 
 13  tweet_location                9907 non-null   object 
 14  user_timezone                 9820 non-null   object 
dtypes: float64(2), int64(2), object(11)
memory usage: 1.7+ MB

In [7]:  tweet_df.columns

Out[7]: Index(['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'],
      dtype='object')

In [8]:  tweet_df.shape

Out[8]: (14640, 15)
```

TEXT CLEANING:

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
stopwords_list = stopwords.words('English')
clean_messages = []
for i in range(tweet_df.shape[0]):
    clean_text = ''
    current_text = word_tokenize(tweet_df['text'].values[i])
    for word in current_text:
        if word == '@' or word == 'VirginAmerica' or word == 'United' or word ==
            'SouthwestAir' or word == 'USAirways' or word == 'americanair' or
            word==        'AmericanAir' or word == 'jetblue' or word == 'delta' or
word ==                  'Delta':
            continue
        if not word in stopwords_list:
            clean_text = clean_text + ' ' + word

    clean_messages.append(clean_text)
```

```
tweet_df['clean_text']=clean_messages
tweet_df['clean_text']=tweet_df['clean_text'].apply(lambda x:" ".join(word.lower() for word in x.split()))
tweet_df['clean_text']=tweet_df['clean_text'].str.replace('[^\w\s]','')
from nltk.stem.wordnet import WordNetLemmatizer
#nltk.download('wordnet')
wordnet = WordNetLemmatizer()
tweet_df['clean_text']=tweet_df['clean_text'].apply(lambda x:'
    '.join(wordnet.lemmatize(word) for word in x.split()))
```

```
In [25]: tweet_df[['text','clean_text']]
```

Out[25]:

	text	clean_text
0	@VirginAmerica What @dhepburn said.	what dhepburn said
1	@VirginAmerica plus you've added commercials t...	plus ve added commercial experience tacky
2	@VirginAmerica I didn't today... Must mean I n...	i nt today must mean i need take another trip
3	@VirginAmerica it's really aggressive to blast...	s really aggressive blast obnoxious entertainm...
4	@VirginAmerica and it's a really big bad thing...	s really big bad thing
...	...	...
14635	@AmericanAir thank you we got on a different f...	thank got different flight chicago
14636	@AmericanAir leaving over 20 minutes Late Flig...	leaving 20 minute late flight no warning commu...
14637	@AmericanAir Please bring American Airlines to...	please bring american airline blackberry10
14638	@AmericanAir you have my money, you change my ...	money change flight nt answer phone any sugges...
14639	@AmericanAir we have 8 ppl so we need 2 know h...	8 ppl need 2 know many seat next flight plz pu...

14640 rows × 2 columns

TEXT DATA IS CLEAN

DATA CLEANING:

4) tweet_df1=tweet_df[['tweet_id','airline_sentiment',
   'airline_sentiment_confidence','airline','clean_text']]

LABEL ENCODING:

```
In [31]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()

In [32]: tweet_df1['airline']=le.fit_transform(tweet_df1['airline'])

In [33]: tweet_df1.head()
```

Out[33]:

	tweet_id	airline_sentiment	airline_sentiment_confidence	airline	clean_text
0	570306133677760513	neutral	1.0000	5	what dhepburn said
1	570301130888122368	positive	0.3486	5	plus ve added commercial experience tacky
2	570301083672813571	neutral	0.6837	5	i nt today must mean i need take another trip
3	570301031407624196	negative	1.0000	5	s really aggressive blast obnoxious entertainm...
4	570300817074462722	negative	1.0000	5	s really big bad thing

QUESTIONS AND ANSWERS:

1.What are the most common words used by people who have taken the airline 'United'?

In [27]:
```
airline_df=tweet_df.loc[(tweet_df['airline']=='United')]
```

In [28]:
```
import re
list1=[]
for i in airline_df['text']:
     list1.append(re.sub(r"[\d,@\'?\.$%_!#^&*)(-+=:;,./?|]", "", i, flags=re.I))
s1=''.join(list1)
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
AI_tokens=word_tokenize(s1)
char_list=['http','lol','-','÷','[',']','#']
AI_tokens= [ele for ele in AI_tokens if all(ch not in ele for ch in char_list)]

from nltk.probability import FreqDist
fdist=FreqDist()
for word in AI_tokens:
    fdist[word.lower()]+=1
print(fdist)
fdist1=fdist.most_common(1)
fdist1
```

```
<FreqDist with 7163 samples and 64061 outcomes>
```

Out[28]:
```
[('to', 2240)]
```

2.What is the most common usertimezone who have taken the airine 'USAirways'?

In [29]:
```
usertimezone_df=tweet_df.loc[(tweet_df['airline']=='Virgin America')]
```

In [30]:
```
from nltk.probability import FreqDist
fdist=FreqDist()
for i in usertimezone_df['user_timezone']:
    fdist[i]+=1
```

In [31]:
```
fdist1=fdist.most_common(1)
```

In [32]:
```
fdist1
```

Out[32]:
```
[('Pacific Time (US & Canada)', 126)]
```

(Using Tokenzied words to select most common word in 1st question
Going through selected rows to find most common user timezone in 2nd question)

TfidfVectorizer and hstack:

TfidfVectorizer:

Convert a collection of raw documents to a matrix of TF-IDF features.

hstack:

Here,HSTACK is used to make the sparse matrix(array) and the other columns into a single array for machine learning,fitting and calculating accuracy score.

In [34]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from scipy.sparse import hstack
vectorizer = TfidfVectorizer(min_df=5)
X_tfidf = vectorizer.fit_transform(tweet_df1['clean_text'])
```

In CountVectorizer we only count the number of times a word appears in the document which results in biasing in favour of most frequent words. this ends up in ignoring rare words which could have helped is in processing our data more efficiently.

To overcome this , we use TfidfVectorizer .

In TfidfVectorizer we consider overall document weightage of a word. It helps us in dealing with most frequent words. Using it we can penalize them. TfidfVectorizer weights the word counts by a measure of how often they appear in the documents.

In [35]:
```
X=hstack([tweet_df1[['tweet_id','airline_sentiment_confidence','airline']],X_tfidf])
```

HERE,WE USE HSTACK TO COMBINE TEXT COLUMN AND OTHER COLUMNS AS OUR INDEPENDENT VARIABLES.

MACHINE LEARNING ALGORITHMS:

LOGISTIC REGRESSION:

```
In [36]: from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression
         logreg=LogisticRegression()
         Y=tweet_df1['airline_sentiment']
         X_train, X_test, Y_train, Y_test = train_test_split(X,Y)
         logreg.fit(X_train,Y_train)
         y_pred=logreg.predict(X_test)
         logreg.score(X_test,Y_test)

Out[36]: 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)

Out[36]: 0.6240437158469946
```

SUPPORT VECTOR CLASSIFIER(SVC):

```
In [37]: from sklearn.svm import SVC
         svc=SVC()
         svc.fit(X_train,Y_train)
         y_pred=svc.predict(X_test)

Out[37]: SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [38]: from sklearn.metrics import accuracy_score

In [39]: accuracy_score(Y_test,y_pred)
Out[39]: 0.6240437158469946
```

DECISION TREE CLASSIFIER:

```
In [40]: from sklearn.tree import DecisionTreeClassifier
         clf = DecisionTreeClassifier()
         clf.fit(X_train,Y_train)

Out[40]: 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')

In [41]: y_pred=clf.predict(X_test)

In [42]: accuracy_score(Y_test,y_pred)
Out[42]: 0.6740437158469945
```

NAIVE BAYES:

```
In [43]:  from sklearn.naive_bayes import MultinomialNB
          X_train, X_test, Y_train, Y_test = train_test_split(X,Y)
          nb = MultinomialNB()
          nb.fit(X_train, Y_train)
          y_pred = nb.predict(X_test)
          from sklearn.metrics import accuracy_score
          accuracy_score(y_pred, Y_test)

Out[43]: MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)

Out[43]: 0.7502732240437159
```

RANDOM FOREST CLASSIFIER:

```
In [44]:  from sklearn.ensemble import RandomForestClassifier
          clf=RandomForestClassifier(n_estimators=100)
          clf.fit(X_train,Y_train)
          y_pred=clf.predict(X_test)

Out[44]: 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=None,
                                verbose=0, warm_start=False)

In [45]:  accuracy_score(Y_test,y_pred)

Out[45]: 0.7519125683060109
```

ENSEMBLE MACHINE LEARNING MODELLING:

Ensemble learning is a machine learning paradigm where multiple models (often called “weak learners”) are trained to solve the same problem and combined to get better results. The main hypothesis is that when weak models are correctly combined we can obtain more accurate and/or robust models.

Some ensemble methods are:
RANDOM FOREST (as shown above)
BAGGING
BOOSTING

BAGGING

A **Bagging classifier** is an ensemble meta-estimator that fits base classifiers each on random subsets of the original dataset and then aggregate their individual predictions (either by voting or by averaging) to form a final prediction. Such a meta-estimator can typically be used as a way to reduce the variance of a black-box estimator (e.g., a decision tree), by introducing randomization into its construction procedure and then making an ensemble out of it.

```
In [46]: from sklearn.ensemble import BaggingClassifier
```

```
In [47]: bg=BaggingClassifier(RandomForestClassifier(),max_samples=0.5,max_features=1.0,n_estimators=10)
         bg.fit(X_train,Y_train)
```

```
Out[47]: BaggingClassifier(base_estimator=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=None,
                                                                  verbose=0,
                                                                  warm_start=False),
                  bootstrap=True, bootstrap_features=False, max_features=1.0,
                  max_samples=0.5, n_estimators=10, n_jobs=None,
                  oob_score=False, random_state=None, verbose=0,
                  warm_start=False)
```

```
In [48]: bg.score(X_test,Y_test)
```

```
Out[48]: 0.742896174863388
```

BOOSTING:

Boosting is an ensemble modeling technique which attempts to build a strong classifier from the number of weak classifiers. It is done building a model by using weak models in series. Firstly, a model is built from the training data. Then the second model is built which tries to correct the errors present in the first model. This procedure is continued and models are added until either the complete training data set is predicted correctly or the maximum number of models are added.

```
In [49]: from sklearn.ensemble import AdaBoostClassifier
```

```
In [50]: adb=AdaBoostClassifier(RandomForestClassifier(),n_estimators=5,learning_rate=1)
         adb.fit(X_train,Y_train)
```

```
Out[50]: AdaBoostClassifier(algorithm='SAMME.R',
                   base_estimator=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=None,
                                                         verbose=0,
                                                         warm_start=False),
                   learning_rate=1, n_estimators=5, random_state=None)
```

```
In [51]: adb.score(X_test,Y_test)
```

```
Out[51]: 0.762568306010929
```

SUMMARY:

We are using various classification algorithms to find accuracy scores and observe which algorithm gives the best accuracy score.We add the text column also as our independent variable through the help of TfidfVectorizer Which converts text into a sparse matrix which is helpful in machine learning.

RANDOM FOREST ALGORITHM suits the best for the **given dataset.** **BOOSTING** has increased its accuracy score a little higher.

ENSEMBLE MODELLING is a process where multiple diverse models are created to predict an outcome, either by using many different modeling algorithms or using different training data sets.

The ensemble model then aggregates the prediction of each base model and results in once final prediction for the unseen data.

The motivation for using ensemble models is to reduce the generalization error of the prediction. As long as the base models are diverse and independent, the prediction error of the model decreases when the ensemble approach is used.