Sentiment analysis in marketing

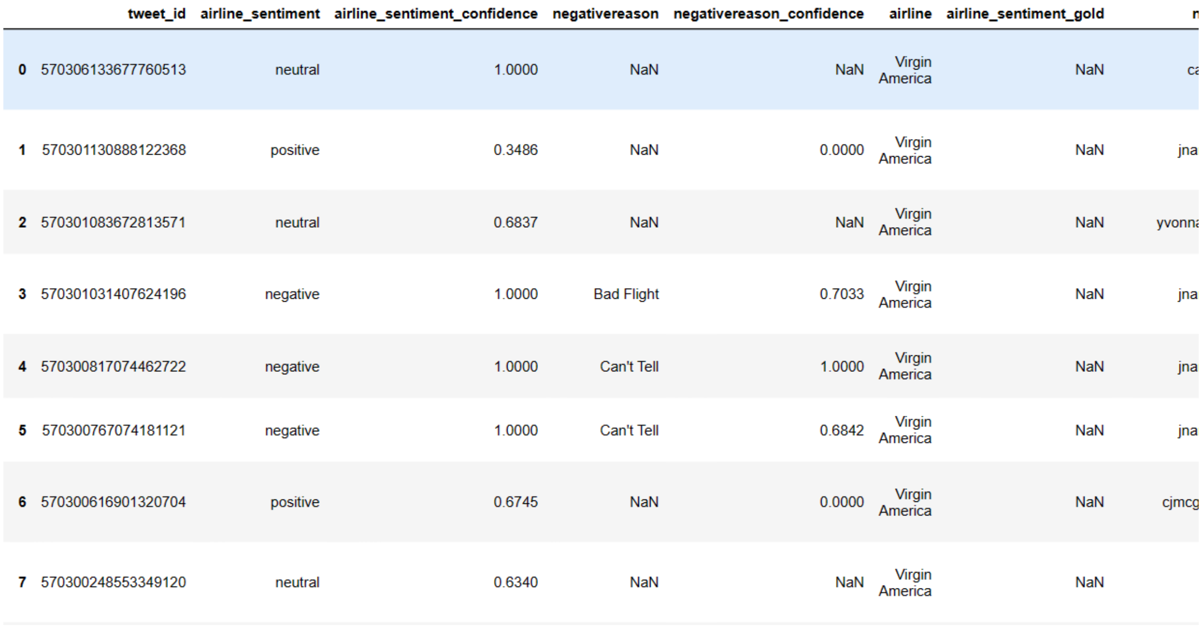
IBM project phase 1

College code:6208

Importing libraries:

```
#!/usr/bin/env python

# coding: utf-8


#  Sentimental Analysis in marketting


# In[ ]:



get_ipython().system('pip install nltk')



# ### Importing Libraries
# In[ ]:


import numpy as np
import pandas as pd
import re #Regular expressions
import nltk
import matplotlib.pyplot as plt

from nltk.corpus import stopwords

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split
```

```
# ### Load Dataset from Local Directory

# In[ ]:


from google.colab import files
uploaded = files.upload()


# ### Importing Dataset

# In[ ]:


dataset = pd.read_csv('dataset.csv')
dataset.head(20)
```



	tweet_id	airline_sentiment	airline_sentiment_confidence	negativereason	negativereason_confidence	airline	airline_sentiment_gold	n
0	570306133677760513	neutral	1.0000	NaN	NaN	Virgin America	NaN	ca
1	570301130888122368	positive	0.3486	NaN	0.0000	Virgin America	NaN	jna
2	570301083672813571	neutral	0.6837	NaN	NaN	Virgin America	NaN	yvonna
3	570301031407624196	negative	1.0000	Bad Flight	0.7033	Virgin America	NaN	jna
4	570300817074462722	negative	1.0000	Can't Tell	1.0000	Virgin America	NaN	jna
5	570300767074181121	negative	1.0000	Can't Tell	0.6842	Virgin America	NaN	jna
6	570300616901320704	positive	0.6745	NaN	0.0000	Virgin America	NaN	cjmcg
7	570300248553349120	neutral	0.6340	NaN	NaN	Virgin America	NaN	

In[ ]:

print(dataset.shape)

Features Extraction:**bold text**

In[ ]:

```
import nltk
nltk.download('wordnet')


# In[ ]:


import nltk
nltk.download('punkt')


# In[ ]:


from sklearn.feature_extraction.text import CountVectorizer
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import string

# Initialize a WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
    # Tokenize the text
    words = word_tokenize(text)

    # Remove punctuation and convert to lower case
    words = [word.lower() for word in words if word.isalpha()]

    # Remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [word for word in words if word not in stop_words]
    # Lemmatize the words
    words = [lemmatizer.lemmatize(word) for word in words]

    return words

# Initialize a CountVectorizer with the custom tokenizer
vectorizer = CountVectorizer(tokenizer=preprocess_text)

# Assume we have a list of texts
texts =["text tweet_coord"]

# Learn the vocabulary and transform the data into a document-term matrix
X = vectorizer.fit_transform(texts)
```

```
# In[ ]:
```

```
print(X)
```

```
# In[ ]:
```

```
dataset.isna().sum()
```

out put:

```
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
```

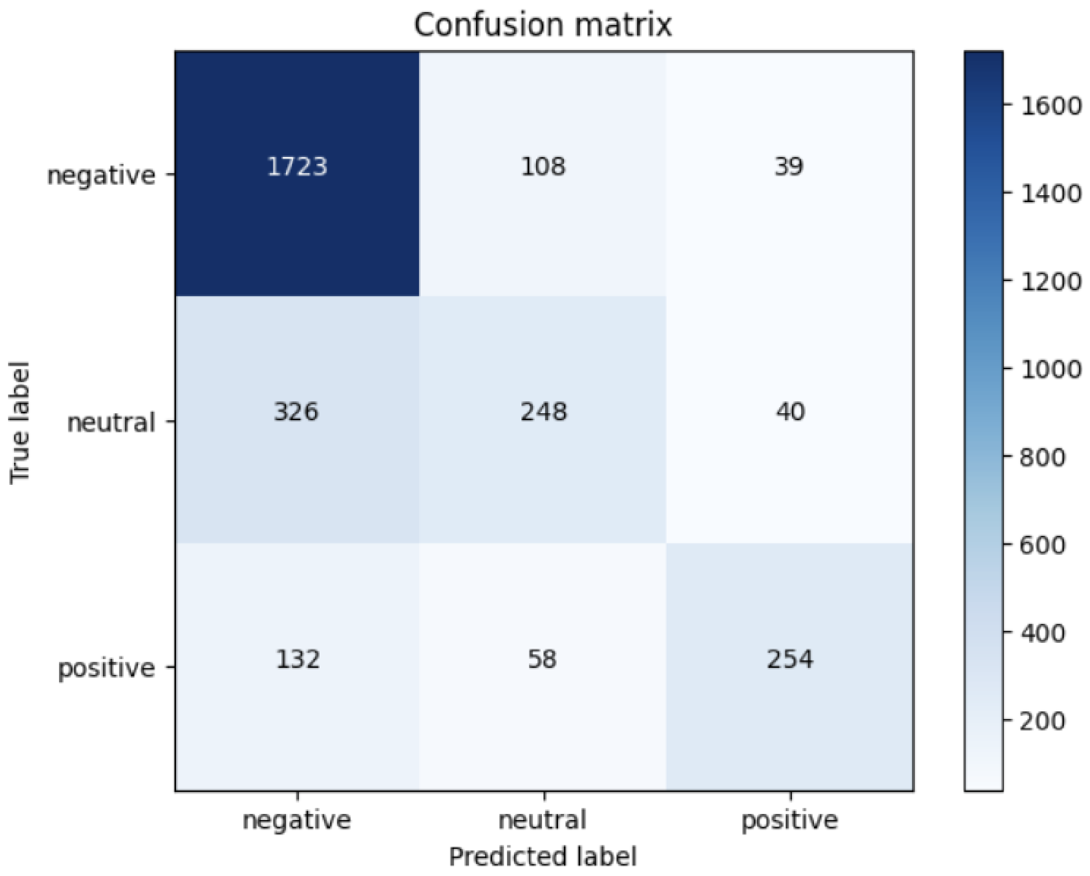
```
# In[ ]:


dataset.describe()


# In[ ]:


dataset.dtypes


# ###Segregating Dataset into Input & Output

# In[ ]:
```

```
features = dataset.iloc[:, 10].values
labels = dataset.iloc[:, 1].values
print(labels)


# ###Removing the Special Character

# In[ ]:


processed_features = []

for sentence in range(0, len(features)):
    # Remove all the special characters
    processed_feature = re.sub(r'\W', ' ',
str(features[sentence]))

    # remove all single characters
    processed_feature= re.sub(r'\s+[a-zA-Z]\s+', ' ',
processed_feature)

    # Remove single characters from the start
    processed_feature = re.sub(r'\^[a-zA-Z]\s+', ' ',
processed_feature)

    # Substituting multiple spaces with single space
    processed_feature = re.sub(r'\s+', ' ',
processed_feature, flags=re.I)

    # Removing prefixed 'b'
    processed_feature = re.sub(r'^b\s+', '',
processed_feature)

    # Converting to Lowercase
    processed_feature = processed_feature.lower()

    processed_features.append(processed_feature)
```

```
# ###Feature Extraction from text
#


# In[ ]:



nltk.download('stopwords')
vectorizer = TfidfVectorizer (max_features=2500, min_df=7, max_df=0.8, stop_words=stopwords.words('english'))
processed_features = vectorizer.fit_transform(processed_features).toarray()
print(processed_features)



# ###Splitting Dataset into Train & Test


# In[ ]:
```

```
X_train, X_test, y_train, y_test =
train_test_split(processed_features, labels,
test_size=0.2, random_state=0)


# ###Loading Random Forest Algorithm

# In[ ]:


text_classifier =
RandomForestClassifier(n_estimators=200,
random_state=0)
text_classifier.fit(X_train, y_train)


# ###Predicting the Test data with Trained Model
# In[ ]:


predictions = text_classifier.predict(X_test)


# In[ ]:


print(X_test)


# ###Score of the Model

# In[ ]:


print(accuracy_score(y_test, predictions))
```

```
# ###Confusion Matrix

# In[ ]:



from sklearn import metrics
import itertools
def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):

    plt.imshow(cm, interpolation='nearest',
cmap=cmap)
    plt.title(title)
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes)
    plt.yticks(tick_marks, classes)

    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]),
range(cm.shape[1])):
        plt.text(j, i, cm[i, j],
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else
"black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

cm = metrics.confusion_matrix(y_test,
predictions, labels=['negative', 'neutral',
'positive'])
plot_confusion_matrix(cm, classes=['negative',
'neutral', 'positive'])
```


Confusion matrix
True label
negative
neutral
positive
1723
108
39
326
248
40
132
58
254
Predicted label
negative
neutral
positive
1600
1400
1200
1000
800
600
400
200