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	negati
2767	568911315026063361	negative	1.0	Late Flight	

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	negati
2767	568911315026063361	negative	1.0	Late Flight	

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_nu
m', 'char_length_original', 'char_length_user', 'mentions_num']].describe()
```

Out[16]:

	emojis_num	emoticons_num	hashtags_num	numbers_num	char_length_original	char_l
count	14640.000000	14640.000000	14640.000000	14640.000000	14640.000000	14
mean	0.066667	0.019262	0.238525	0.429372	103.822063	
std	0.612111	0.140400	0.654195	0.741321	36.277339	
min	0.000000	0.000000	0.000000	0.000000	12.000000	
25%	0.000000	0.000000	0.000000	0.000000	77.000000	
50%	0.000000	0.000000	0.000000	0.000000	114.000000	
75%	0.000000	0.000000	0.000000	1.000000	136.000000	
max	40.000000	3.000000	8.000000	7.000000	186.000000	

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', 'cha
r_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_nu
m', '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'].asty
pe(int)
df_tweets_cleaned['hashtags_flag'] = df_tweets_cleaned['hashtags_flag'].astype
(int)
df_tweets_cleaned['numbers_flag'] = df_tweets_cleaned['numbers_flag'].astype(i
nt)
df_tweets_cleaned['retweet_flag'] = df_tweets_cleaned['retweet_flag'].astype(i
nt)
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', 'binar
y_response_variable'] = False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'positive', 'bina
ry_response_variable'] = False
df_tweets_cleaned.loc[df_tweets_cleaned.airline_sentiment == 'negative', 'bina
ry_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	nega
0	570306133677760513	neutral	1.0000		
1	570301130888122368	positive	0.3486		
2	570301083672813571	neutral	0.6837		
3	570301031407624196	negative	1.0000	Bad Flight	
4	570300817074462722	negative	1.0000	Can't Tell	
...	...	...	...	...	
14635	569587686496825344	positive	0.3487		
14636	569587371693355008	negative	1.0000	Customer Service Issue	
14637	569587242672398336	neutral	1.0000		
14638	569587188687634433	negative	1.0000	Customer Service Issue	
14639	569587140490866689	neutral	0.6771		

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_emoticon
s',
       '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_var
iable']
Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_var
iable']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, rand
om_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, binary=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_mod
el, 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_ave
rage_tweet_vector(text, w2v_gensim_model, num_dimensions))
X_test['tweet_vector'] = X_test.lemmas_list.apply(lambda text: calculate_avera
ge_tweet_vector(text, w2v_gensim_model, num_dimensions))
```

In [30]:
```
df_train_tweet_vector = pd.DataFrame(list(X_train['tweet_vector']), index=X_tr
ain.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 ne
ed 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_vec
tor']
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_li
st']
```

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_var
iable']
Y = df_tweets_cleaned.loc[:, df_tweets_cleaned.columns == 'binary_response_var
iable']

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

Generate Tweet vectors

In [37]:
```
import gensim

custom_w2v_gensim_model_path = '..\milestone1\Patrick\gensim_word2vec_google_s
ubset.bin'
w2v_gensim_model = gensim.models.KeyedVectors.load_word2vec_format(custom_w2v_
gensim_model_path, binary=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, w2v_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 [ ]: