What is Sentiment Analysis?

Sentiment analysis is a process of identifying an attitude of the author on a topic that is being written about.

Table of contents

1. Load the libraries
2. Load Dataset
3. EDA
4. Preprocessing Tweet Text
5. Featurization
    1. Bag-of-Words
    2. TF-IDF
    3. Word2vec
6. Resample
    1. Upsampling BOW
    2. Upsampling TF-IDF
    3. Upsampling word2vec
7. Split Dataset
8. Model Selection
    1. KNN
9. Summary

Load the libraries

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_i

Enter your authorization code:
··········
Mounted at /content/drive
```

```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import nltk
from nltk.stem import PorterStemmer,WordNetLemmatizer
from wordcloud import WordCloud
import re
import gensim
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer,TfidfVectorizer
from sklearn.utils import resample
from sklearn.metrics import accuracy_score,f1_score
```

Load Dataset

```
df=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Twitter/train.csv.zip')
test=pd.read_csv('/content/drive/My Drive/Colab Notebooks/Twitter/test.csv.zip')
df.head(4)
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...

EDA

```
df.info()
```

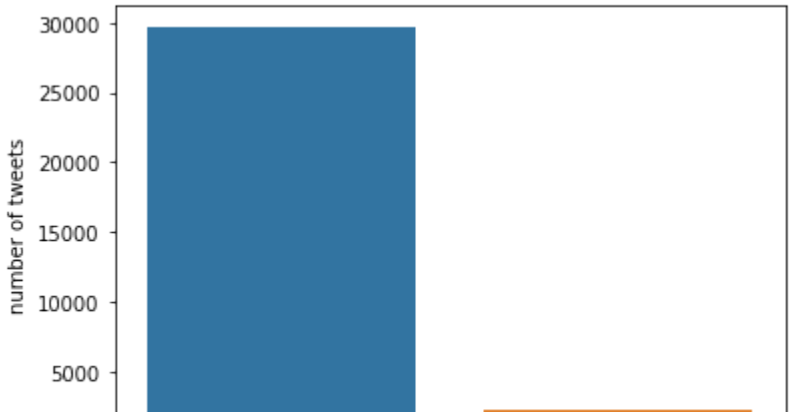
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31962 entries, 0 to 31961
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      31962 non-null  int64 
 1   label   31962 non-null  int64 
 2   tweet   31962 non-null  object
dtypes: int64(2), object(1)
memory usage: 749.2+ KB
```

```
print('shape of train dataset',df.shape)
df.label.value_counts()
```

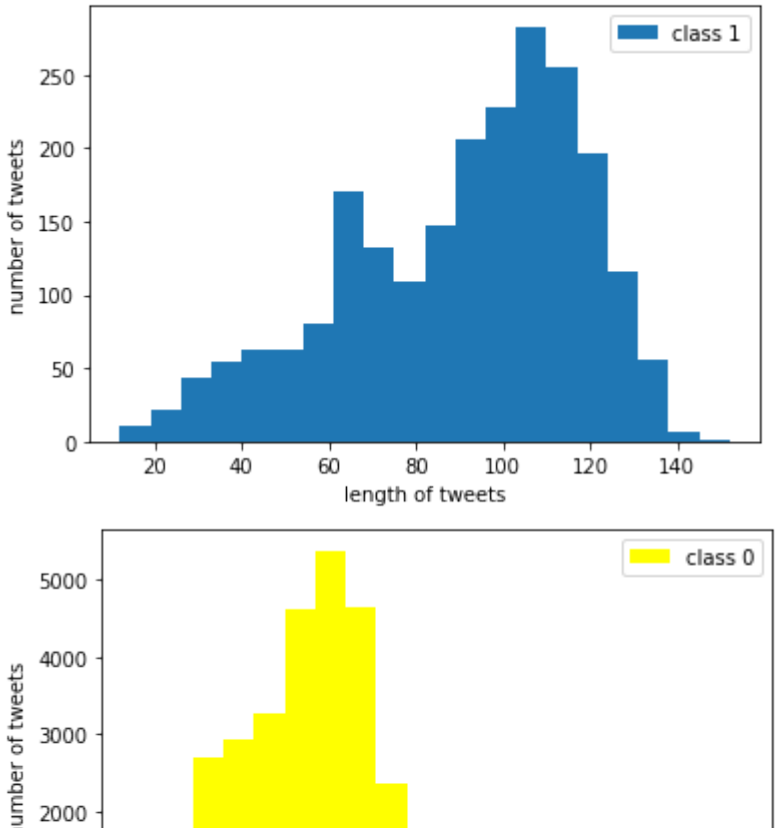
```
shape of train dataset (31962, 3)
```

```
0    29720
1     2242
Name: label, dtype: int64
```

```
#
sns.countplot(df.label,)
plt.xlabel('class label')
plt.ylabel('number of tweets')
plt.show()
```




```
plt.hist(df[df['label']==1].tweet.str.len(),bins=20,label='class 1')
plt.legend()
plt.xlabel('length of tweets')
plt.ylabel('number of tweets')
plt.show()
plt.hist(df[df['label']==0].tweet.str.len(),color='yellow',bins=20,label='class 0')
plt.legend()
plt.xlabel('length of tweets')
plt.ylabel('number of tweets')
plt.show()
```



Preprocessing Tweet Text

1. Removing Twitter Handles (@user)
2. Removing urls from text
3. Removing Punctuations, Numbers, and Special Characters
4. Convert the word to lowercase
5. Remove Stopwords
6. Stemming the word
7. Lemmatization

After which we collect the words used to describe positive and negative reviews

```
text=df['tweet'].values.tolist()
text_test=test['tweet'].values.tolist()
```

```
text+=text_test
print(len(text))
```

```
49159
```

```
import nltk
stopword=nltk.corpus.stopwords.words('english')
stopword.remove('not')
for index,text_ in enumerate(text):
    text_=re.sub(r'@[\w]*','',text_) #Removing Twitter Handles (@user)
    text_=re.sub(r'http/S+','',text_) #Removing urls from text
    text_=re.sub(r'[^A-Za-z#]',' ',text_) #Removing Punctuations, Numbers, and Special Cha
    text_=" ".join(i.lower() for i in text_.split() if i.lower() not in stopword) #Removin
    text[index]=text_
```

```
#Stemming the word
pt=PorterStemmer()
wordnet=WordNetLemmatizer()
for index,text_ in enumerate(text):
    text_=" ".join(pt.stem(i) for i in text_.split())
    text_=" ".join(wordnet.lemmatize(i) for i in text_.split())
    text[index]=text_
```

```
df['preprocess_tweet']=text[:len(df)]
df['length_tweet']=df['preprocess_tweet'].str.len()
test['preprocess_tweet']=text[len(df):]
df.head()
```

	id	label	tweet	preprocess_tweet	length_tweet
0	1	0	@user when a father is dysfunctional and is s...	father dysfunct selfish drag kid dysfunct #run	46
1	2	0	@user @user thanks for #lyft credit i can't us...	thank #lyft credit use caus offer wheelchair v...	73
2	3	0	bihday your majesty	bihday majesti	14
3	4	0	#model i love u take with u all the time in ...	#model love u take u time ur	28

Featurization

BOW

```
train=df.copy()
train.drop(columns=['id','tweet','preprocess_tweet'],inplace=True)
```

```
bow=CountVectorizer( min_df=2, max_features=1000)
bow.fit(df['preprocess_tweet'])
bow_df=bow.transform(df['preprocess_tweet']).toarray()
print('feature name==',bow.get_feature_names()[:10])
print('number of uniqe words',bow_df.shape[1])
print('shape',bow_df.shape)
bow_train=pd.DataFrame(bow_df)
bow_train['length_tweet']=df['length_tweet']
bow_train['label']=df['label']
bow_train.head()
```

```
feature name== ['abl', 'absolut', 'accept', 'account', 'act', 'action', 'actor'
number of uniqe words 1000
shape (31962, 1000)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

5 rows × 1002 columns

TF-IDF Features (Bi-Grams)

```
tfidf=TfidfVectorizer(ngram_range=(1, 2),min_df=2,max_features=1000)
tfidf.fit(df['preprocess_tweet'])
tfidf_df=tfidf.transform(df['preprocess_tweet']).toarray()
print('number of uniqe words',bow_df.shape[1])
print('shape',tfidf_df.shape)
tfidf_train=pd.DataFrame(tfidf_df)
tfidf_train['length_tweet']=df['length_tweet']
tfidf_train['label']=df['label']
tfidf_train.head()
```

```
number of uniqe words 1000
shape (31962, 1000)
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

5 rows × 1002 columns

Word2vec

size: The number of dimensions of the embeddings and the default is 100.

window: The maximum distance between a target word and words around the target word. The default window is 5.

min_count: The minimum count of words to consider when training the model; words with occurrence less than this count will be ignored. The default for min_count is 5.

workers: The number of partitions during training and the default workers is 3.

sg: The training algorithm, either CBOW(0) or skip gram(1). The default training algorithm is CBOW.

```
tokenize=df['preprocess_tweet'].apply(lambda x: x.split())
w2vec_model=gensim.models.Word2Vec(tokenize,min_count = 1, size = 100, window = 5, sg = 1)
w2vec_model.train(tokenize,total_examples= len(df['preprocess_tweet']),epochs=20)
```

```
(4813662, 5011220)
```

```
w2vec_model.most_similar('father')
```

```
[('#fathersday', 0.7850511074066162),
 ('dad', 0.7521795034408569),
 ('#dad', 0.7398676872253418),
 ('#father', 0.7331924438476562),
 ('fathersday', 0.7148988246917725),
 ('papa', 0.710792064666748),
 ('#hackney', 0.6768671274185181),
 ('hrithik', 0.6764564514160156),
 ('#daddi', 0.6736965179443359),
 ('#felizdiadelpadr', 0.6735095381736755)]
```

```
w2v_words = list(w2vec_model.wv.vocab)
print("number of words that occured minimum 5 times ",len(w2v_words))
print("sample words ", w2v_words[0:50])
```

```
number of words that occured minimum 5 times  36842
sample words  ['father', 'dysfunct', 'selfish', 'drag', 'kid', '#run', 'thank',
```

```
vector=[]
from tqdm import tqdm
for sent in tqdm(tokenize):
  sent_vec=np.zeros(100)
  count =0
  for word in sent:
    if word in w2v_words:
      vec = w2vec_model.wv[word]
      sent_vec += vec
      count += 1
  if count != 0:
    sent_vec /= count #normalize
  vector.append(sent_vec)
print(len(vector))
print(len(vector[0]))
```

```
100%|██████████| 31962/31962 [00:23<00:00, 1376.07it/s]31962
100
```

```
#example
l='father dysfunct selfish drag kid dysfunct'
count=0
vcc=np.zeros(100)
for word in l:
  if word in w2v_words:
    v=w2vec_model.wv[word]
    vcc+=v
    count+=1
vcc
```

```
array([  0.91146523,   0.19254029,  -5.33413239,   1.77129817,
         3.93843668,  -2.07992977,  -0.40524426,  11.83810888,
        -4.79120433,  -1.82969081,   4.73461972,   3.30368719,
       -13.09859578,  15.80826025,   1.53046007, -12.68571611,
         8.36263441,   6.29655315,   0.92606603,   1.92379364,
        -4.13300864,   0.88979801,  -2.32401064,   6.59359596,
        -3.36224241,   7.18753285,   8.49213998,   4.75367435,
        21.37358405,  -5.26452677,   2.99436364,  -3.46574163,
        -8.2754163 ,  -3.92128712,   7.66723774,   3.03709563,
        -4.56741019,   2.67773311,   2.44705757,   7.84918264,
       -10.29537304, -10.46700041,   2.74154223,  -8.12427359,
        13.18823193,  12.24374422,  -1.87443534,  15.19453245,
         5.66711175,  16.97487455, -13.27168938,  -9.83208083,
         7.40258518,  -6.98420832,  -7.12414303,   8.19903404,
       -10.53523948,   8.87091764,  -7.79694048,   4.03617226,
         8.59358842,   6.62909203,  10.59675463,  -5.33136314,
       -10.46381125,   6.69054185, -11.99697267,   7.03941123,
        -8.07251544,   3.54266782,   2.56590302,  -8.83850726,
        -7.12371032,  -4.35939002,  -7.29442378, -14.31221112,
```

```
       -14.45862933,  -5.46173335,   2.38489718,  -2.84350388,
         8.20834357,   6.23576167,  -0.47916248,  -8.48871913,
         4.90017756,   6.23322728, -15.76709158,  -2.11499437,
       -19.72047988,   7.70427328,  -9.13389167,  10.27496818,
        -4.77674974,  -1.94897728,  -7.12096034,  -4.71564086,
        -0.87702771,   3.15800697, -15.20042363,  -9.61333981])
```

```
print('number of uniqe words',len(vector[1]))
w2v_train=pd.DataFrame(vector)
w2v_train['length_tweet']=df['length_tweet']
w2v_train['label']=df['label']
w2v_train.head()
```

```
number of uniqe words 100
```

	0	1	2	3	4	5	6	7	
0	-0.258498	-0.363275	-0.182613	0.114784	0.018073	-0.063409	0.340427	0.094704	0.1
1	-0.053422	-0.258770	0.056477	0.497901	-0.113738	0.015618	-0.041178	0.194600	0.0
2	-0.567869	-0.360663	0.506734	0.320363	-0.706200	0.729753	0.143181	-0.062848	-0.7
3	-0.417183	-0.274130	0.264695	-0.187070	0.456043	-0.142716	-0.061042	0.621747	-0.6
4	-0.115846	-0.392109	-0.506213	0.475562	-0.069108	0.231304	-0.098725	0.250521	0.1

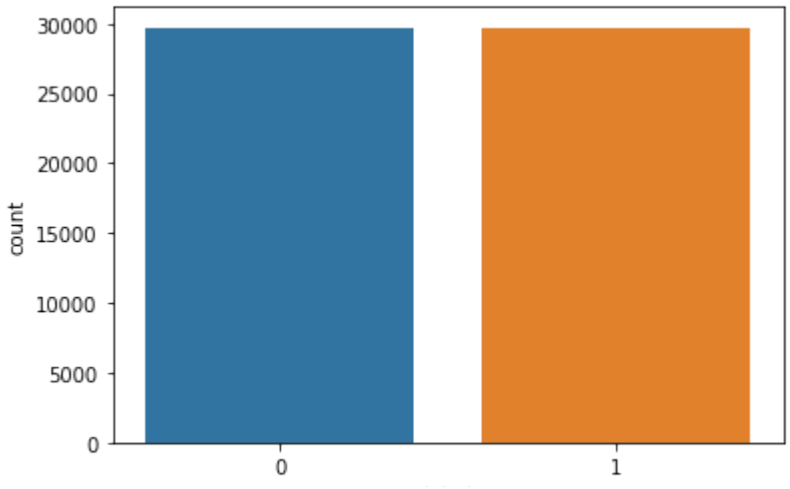
5 rows × 102 columns

Resample

Upsampling BOW

```
major_class_0,major_class_1=bow_train.label.value_counts()
df_major=bow_train[bow_train['label']==0]
df_minor=bow_train[bow_train['label']==1]
df_minor_upsampled = resample(df_minor,
                              replace=True,     # sample with replacement
                              n_samples=major_class_0)
df_bow_upsampled = pd.concat([df_major, df_minor_upsampled])
print('shape',df_bow_upsampled.shape)
sns.countplot(df_bow_upsampled.label)
```

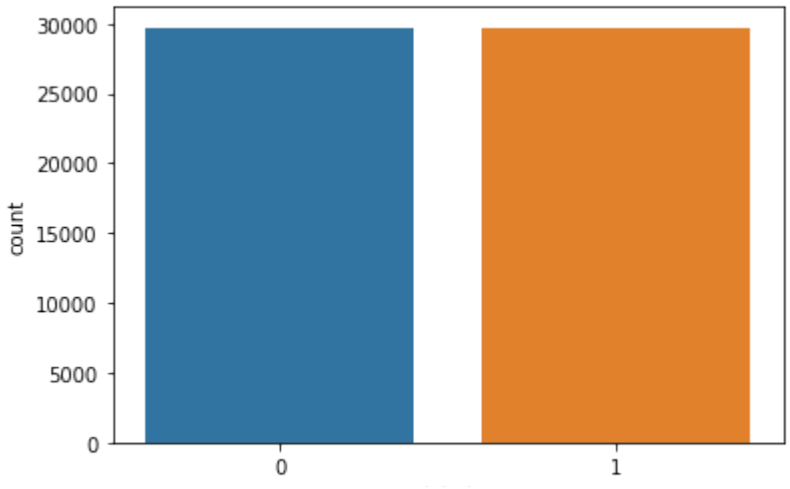
```
shape (59440, 1002)
<matplotlib.axes._subplots.AxesSubplot at 0x7f16260a2f60>
```



Upsampling TF-IDF

```
major_class_0,major_class_1=tfidf_train.label.value_counts()
df_major=tfidf_train[tfidf_train['label']==0]
df_minor=tfidf_train[tfidf_train['label']==1]
df_minor_upsampled = resample(df_minor,
                              replace=True,     # sample with replacement
                              n_samples=major_class_0)
df_tfidf_upsampled = pd.concat([df_major, df_minor_upsampled])
print('shape',df_tfidf_upsampled.shape)
sns.countplot(df_tfidf_upsampled.label)
```

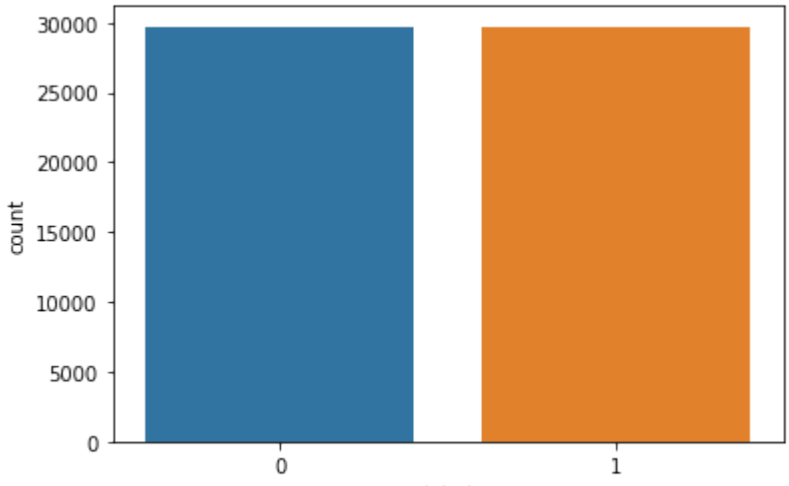
```
shape (59440, 1002)
<matplotlib.axes._subplots.AxesSubplot at 0x7f16260a2668>
```



Upsampling word2vec

```
major_class_0,major_class_1=w2v_train.label.value_counts()
df_major=w2v_train[w2v_train['label']==0]
df_minor=w2v_train[w2v_train['label']==1]
df_minor_upsampled = resample(df_minor,
                                replace=True,     # sample with replacement
                                n_samples=major_class_0)
df_w2v_upsampled = pd.concat([df_major, df_minor_upsampled])
print('shape',df_w2v_upsampled.shape)
sns.countplot(df_w2v_upsampled.label)
```

```
shape (59440, 102)
<matplotlib.axes._subplots.AxesSubplot at 0x7f162166b550>
```



Split Dataset

```
x=df_bow_upsampled.iloc[:,0:-1]
y=df_bow_upsampled['label']
x_train_bow,x_test_bow,y_train_bow,y_test_bow=train_test_split(x,y,test_size=0.2)
```

```
x=df_tfidf_upsampled.iloc[:,0:-1]
y=df_tfidf_upsampled['label']
x_train_tfidf,x_test_tfidf,y_train_tfidf,y_test_tfidf=train_test_split(x,y,test_size=0.2)
```

```
x=df_w2v_upsampled.iloc[:,0:-1]
y=df_w2v_upsampled['label']
x_train_w2v,x_test_w2v,y_train_w2v,y_test_w2v=train_test_split(x,y,test_size=0.2)
```

Model Selection

```
def f1_score_(y_proba,y_test):
  proba = y_proba[:,1] >= 0.3
  proba = proba.astype(np.int)
  return f1_score( proba,y_test)
```

KNN

```
#use Bow
from sklearn.neighbors import KNeighborsClassifier
k=[3,5,7,11]
accuracy=[]
for i in tqdm(k):
  model=KNeighborsClassifier(n_neighbors=i)
  model.fit(x_train_bow,y_train_bow)
  y_pred=model.predict(x_test_bow)
  acc=accuracy_score(y_pred,y_test_bow)
  print('for k=',i,'Accuracy Score',acc)
  accuracy.append(acc)
  y_proba=model.predict_proba(x_test_bow)
  f1_scor=f1_score_(y_proba,y_test_bow)
  print('for k=',i,'f1 score ',f1_scor)
```

```
  0%|          | 0/4 [00:00<?, ?it/s]for k= 3 Accuracy Score 0.8681864064602961
 25%|███       | 1/4 [01:37<04:52, 97.62s/it]for k= 3 f1 score  0.8384620752215
for k= 5 Accuracy Score 0.8405114401076716
 50%|██████    | 2/4 [03:23<03:20, 100.20s/it]for k= 5 f1 score  0.82993291375
for k= 7 Accuracy Score 0.8201547779273217
 75%|█████████ | 3/4 [05:17<01:44, 104.21s/it]for k= 7 f1 score  0.81907490790
for k= 11 Accuracy Score 0.7850773889636609
100%|████████████| 4/4 [07:26<00:00, 111.51s/it]for k= 11 f1 score  0.780025940
```

```
#use tfidf
k=[3,5,11]
accuracy_tfidf=[]
for i in k:
  model=KNeighborsClassifier(n_neighbors=i)
  model.fit(x_train_tfidf,y_train_tfidf)
  y_pred=model.predict(x_test_tfidf)
  acc=accuracy_score(y_pred,y_test_tfidf)
  print('for k=',i,'Accuracy Score',acc)
  accuracy_tfidf.append(acc)
  y_proba=model.predict_proba(x_test_tfidf)
  f1_scor=f1_score_(y_proba,y_test_tfidf)
  print('for k=',i,'f1 score ',f1_scor)
```

```
for k= 3 Accuracy Score 0.856914535666218
for k= 3 f1 score  0.8520794626268402
for k= 5 Accuracy Score 0.84185733512786
for k= 5 f1 score  0.8429833863556027
for k= 11 Accuracy Score 0.8036675639300135
for k= 11 f1 score  0.8097859327217125
```

```
#use word2vec
from sklearn.neighbors import KNeighborsClassifier
k=[3,5,11]
accuracy_w2v=[]
for i in tqdm(k):
  model=KNeighborsClassifier(n_neighbors=i)
  model.fit(x_train_w2v,y_train_w2v)
  y_pred=model.predict(x_test_w2v)
  acc=accuracy_score(y_pred,y_test_w2v)
  print('for k=',i,'Accuracy Score',acc)
  accuracy_w2v.append(acc)
  y_proba=model.predict_proba(x_test_w2v)
  f1_scor=f1_score_(y_proba,y_test_w2v)
  print('for k=',i,'f1 score ',f1_scor)
```

```
  0%|          | 0/3 [00:00<?, ?it/s]for k= 3 Accuracy Score 0.9531460296096904
 33%|███       | 1/3 [00:12<00:25, 12.76s/it]for k= 3 f1 score  0.941370120708
for k= 5 Accuracy Score 0.9362382234185733
 67%|██████    | 2/3 [00:26<00:13, 13.08s/it]for k= 5 f1 score  0.926340146548
```