notebook

July 27, 2023

0.0.1 Sentiment Analysis of IMDB Movie Reviews

Problem Statement : Given 50,000 reviews, our task is to predict whether the data gathered for testing would show positive or negative sentiment.

We will keep some data for testing and use the rest for training.

Importing Libraries

[4]:
```
import pandas as pd
import numpy as np

import re
import string

import gensim
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
from gensim.utils import simple_preprocess
from nltk.stem import WordNetLemmatizer
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from wordcloud import WordCloud

import pickle
```

Importing dataset

[5]:
```
path = 'Data/imdb_data.csv'
df = pd.read_csv(path)
df.head(3)
```

```
[5]:                                                review sentiment
      0  One of the other reviewers has mentioned that …  positive
      1  A wonderful little production. <br /><br />The…  positive
      2  I thought this was a wonderful way to spend ti…  positive
```

```
[6]: df.shape
```

```
[6]: (50000, 2)
```

```
[7]: df.describe()
```

```
[7]:                                                     review sentiment
      count                                               50000     50000
      unique                                              49582         2
      top     Loved today's show!!! It was a variety and not…  positive
      freq                                                    5     25000
```

```
[8]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   review     50000 non-null  object
 1   sentiment  50000 non-null  object
dtypes: object(2)
memory usage: 781.4+ KB
```

Looking for any duplicate or missing values :

```
[5]: df.duplicated().sum()
```

```
[5]: 418
```

```
[6]: df.drop_duplicates(inplace=True)
```

```
[7]: df.isna().sum()
```

```
[7]: review       0
     sentiment    0
     dtype: int64
```

```
[8]: df.shape
```

```
[8]: (49582, 2)
```

Sentiment count :

```
[9]: df['sentiment'].value_counts()
```

```
[9]: positive    24884
     negative    24698
     Name: sentiment, dtype: int64
```

Lowercasing the original sentences :

```
[9]: df['review'] = df['review'].apply(lambda x: x.lower())
     df['review'].sample()
```

```
[9]: 39089    this movie is about three teens who have been …
     Name: review, dtype: object
```

Removing unwanted HTML tags :

```
[10]: def htmltags(txt):
          txt = re.sub(re.compile('<.*?>'), '', txt)
          return txt
```

```
[11]: df['review'].apply(htmltags)
```

```
[11]: 0        one of the other reviewers has mentioned that …
      1        a wonderful little production. the filming tec…
      2        i thought this was a wonderful way to spend ti…
      3        basically there's a family where a little boy …
      4        petter mattei's "love in the time of money" is…
                                      …
      49995    i thought this movie did a down right good job…
      49996    bad plot, bad dialogue, bad acting, idiotic di…
      49997    i am a catholic taught in parochial elementary…
      49998    i'm going to have to disagree with the previou…
      49999    no one expects the star trek movies to be high…
      Name: review, Length: 50000, dtype: object
```

Removing Stopwords :

```
[12]: sw = stopwords.words('english')
      df['review'] = df['review'].apply(lambda x: [i for i in x.split() if i not in␣
       ↪sw]).apply(lambda x: ' '.join(x))

      df.sample()
```

```
[12]:                                                  review sentiment
      43422  like movies morally corrupt characters, much. …  negative
```

Removing URLs :

[13]:

```python
def removeUrl(text):
    p =re.compile(r"https?://\S+|www\.\S+")
    return p.sub(r'', text)

df['review'] = df['review'].apply(removeUrl)
```

Removing Punctuations :

[14]:

```python
punc= string.punctuation
punc
```

[14]:

```
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

[15]:

```python
def removePunc(text):
    return text.translate(str.maketrans('','', punc))

df['review'] = df['review'].apply(removePunc)
```

Lemmatization :

[16]:

```python
lemmatizer = WordNetLemmatizer()

def lem(txt):
    s = []
    sent = word_tokenize(txt)
    for word in sent:
        s.append(lemmatizer.lemmatize(word))

    return ' '.join(s)
```

[19]:

```python
# !unzip /usr/share/nltk_data/corpora/wordnet.zip -d /usr/share/nltk_data/corpora/
```

[17]:

```python
df['review'] = df['review'].apply(lem)
```

Label Encoding :

[21]:

```python
en = LabelEncoder()
y = en.fit_transform(df['sentiment'])
y
```

[21]:

```
array([1, 1, 1, …, 0, 0, 0])
```

[24]:

```python
comment = []
for doc in df['review']:
    raw = sent_tokenize(doc)
    for sent in raw:
        comment.append(simple_preprocess(sent))
```

Converting Word into Vectors using Word2Vec algorithm :

[25]:
```
model = gensim.models.Word2Vec(
    window = 10,
    min_count = 2
)
```

[26]:
```
len(comment)
```

[26]:
```
49582
```

[27]:
```
model.build_vocab(comment)
```

[28]:
```
model.train(comment, total_examples=model.corpus_count, epochs=model.epochs)
```

[28]:
```
(28666106, 30851370)
```

Length of vocabulary (no. of words in vocabulary) :

[29]:
```
len(model.wv.index_to_key)
```

[29]:
```
70860
```

Average Word2Vec : We simply average each word of a document so that the generated document vector is actually a centroid of all words in feature space.

[30]:
```
def document_vector(doc):
    new = [i for i in doc.split() if i in model.wv.index_to_key]
    return np.mean(model.wv[new], axis=0)
```

Example :

[31]:
```
document_vector(df['review'].values[0])
```

[31]:
```
array([ 0.62246203,  0.09984188,  0.04422702, -0.22960517, -0.0772144 ,
       -0.5070311 ,  0.8425616 ,  0.5891773 ,  0.04115144,  0.24194868,
        0.03238557, -0.11320531,  0.00557064,  0.01963594, -0.06383892,
       -0.0078267 , -0.1792575 ,  0.13221012, -0.11584643, -0.29176405,
       -0.33760226, -0.07981902, -0.29290378, -0.3863513 , -0.32592434,
        0.00689173,  0.08336066, -0.22146599,  0.29623184, -0.1229619 ,
        0.8419347 , -0.5328552 ,  0.29248852, -0.5249922 ,  0.45816648,
        0.42495805,  0.38128042,  0.14935799, -0.18571083, -0.3879461 ,
        0.07501417, -0.4529647 , -0.19089752,  0.08730777,  0.3032065 ,
       -0.55529606,  0.1484235 , -0.19394206,  0.18591803, -0.17241104,
        0.1692754 , -0.45281383, -0.24183159,  0.08929349, -0.18101156,
        0.10384404,  0.44697493,  0.01407371,  0.20089446, -0.2572593 ,
        0.12930632, -0.22554857, -0.39358962, -0.28974158, -0.3282461 ,
        0.21016835,  0.5591773 , -0.17272896, -0.2972311 ,  0.02883279,
       -0.41485834, -0.2603172 , -0.20533642,  0.40229616,  0.81266224,
       -0.21213953,  0.01862841,  0.06164954, -0.14734235,  0.4888641 ,
        0.19944413,  0.04785746, -0.51586014, -0.00430945, -0.00297356,
        0.1420607 , -0.16435936,  0.3427634 ,  0.51407593,  0.36339602,
       -0.4927859 ,  0.26370877,  0.22366601, -0.1276373 ,  0.21800439,
       -0.62615436, -0.08236498, -0.02277287, -0.03556763,  0.18901771],
      dtype=float32)
```

Creating Training Dataset :

```
[32]: X = []
      for i in tqdm(df['review'].values):
          X.append(document_vector(i))

      len(X)
```

```
100%|      | 49582/49582 [16:03<00:00, 51.44it/s]
```

```
[32]: 49582
```

```
[33]: X = np.array(X)
      X.shape
```

```
[33]: (49582, 100)
```

```
[34]: X[0]
```

```
[34]: array([ 0.62246203,  0.09984188,  0.04422702, -0.22960517, -0.0772144 ,
       -0.5070311 ,  0.8425616 ,  0.5891773 ,  0.04115144,  0.24194868,
        0.03238557, -0.11320531,  0.00557064,  0.01963594, -0.06383892,
       -0.0078267 , -0.1792575 ,  0.13221012, -0.11584643, -0.29176405,
       -0.33760226, -0.07981902, -0.29290378, -0.3863513 , -0.32592434,
        0.00689173,  0.08336066, -0.22146599,  0.29623184, -0.1229619 ,
        0.8419347 , -0.5328552 ,  0.29248852, -0.5249922 ,  0.45816648,
        0.42495805,  0.38128042,  0.14935799, -0.18571083, -0.3879461 ,
        0.07501417, -0.4529647 , -0.19089752,  0.08730777,  0.3032065 ,
       -0.55529606,  0.1484235 , -0.19394206,  0.18591803, -0.17241104,
        0.1692754 , -0.45281383, -0.24183159,  0.08929349, -0.18101156,
        0.10384404,  0.44697493,  0.01407371,  0.20089446, -0.2572593 ,
        0.12930632, -0.22554857, -0.39358962, -0.28974158, -0.3282461 ,
        0.21016835,  0.5591773 , -0.17272896, -0.2972311 ,  0.02883279,
       -0.41485834, -0.2603172 , -0.20533642,  0.40229616,  0.81266224,
       -0.21213953,  0.01862841,  0.06164954, -0.14734235,  0.4888641 ,
        0.19944413,  0.04785746, -0.51586014, -0.00430945, -0.00297356,
        0.1420607 , -0.16435936,  0.3427634 ,  0.51407593,  0.36339602,
       -0.4927859 ,  0.26370877,  0.22366601, -0.1276373 ,  0.21800439,
       -0.62615436, -0.08236498, -0.02277287, -0.03556763,  0.18901771],
        dtype=float32)
```

Spliting the dataset (80-20%) :

[43]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
  ↪random_state=1, stratify=y)
```

[44]:
```
X_train.shape
```

[44]:
```
(39665, 100)
```

Modelling

[45]:
```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
```

Accuracy of model on test data :

[46]:
```
score = accuracy_score(y_test, y_pred)
```

[47]:
```
score
```

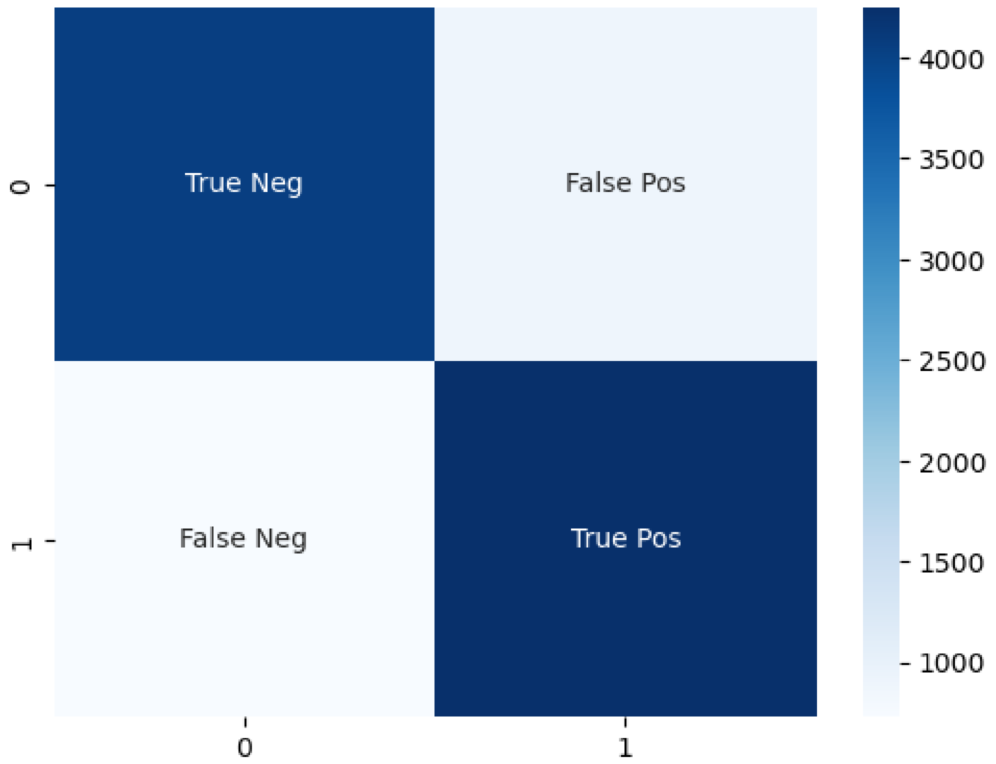
[47]:
```
0.8370474942018755
```

Plotting confusion matrix

[48]:
```
from sklearn.metrics import confusion_matrix
cf = confusion_matrix(y_test, y_pred)
```

[58]:
```
labels = ['True Neg','False Pos','False Neg','True Pos']
labels = np.asarray(labels).reshape(2,2)
sns.heatmap(cf, annot=labels, fmt='', cmap='Blues')
```

[58]:
```
<Axes: >
```



Classification Report :

```
[61]: print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.85      0.82      0.83      4940
           1       0.83      0.85      0.84      4977

    accuracy                           0.84      9917
   macro avg       0.84      0.84      0.84      9917
weighted avg       0.84      0.84      0.84      9917
```

Analysing the Sentiments using WordCloud : WordCloud for Positive Words.

```
[28]: df['sentiment'].head(5)
```

```
[28]: 0    positive
      1    positive
      2    positive
      3    negative
      4    positive
```

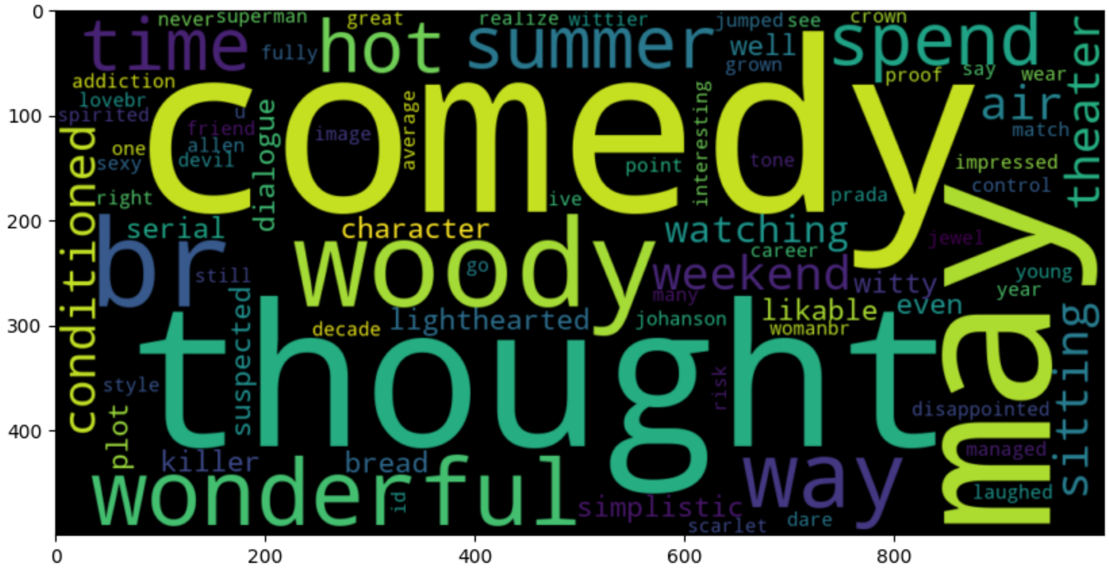
```
Name: sentiment, dtype: object
```

[30]:

```
plt.figure(figsize=(10,10))
text=df["review"][2]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
words=WC.generate(text)
plt.imshow(words,interpolation='bilinear')
plt.show
```

[30]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



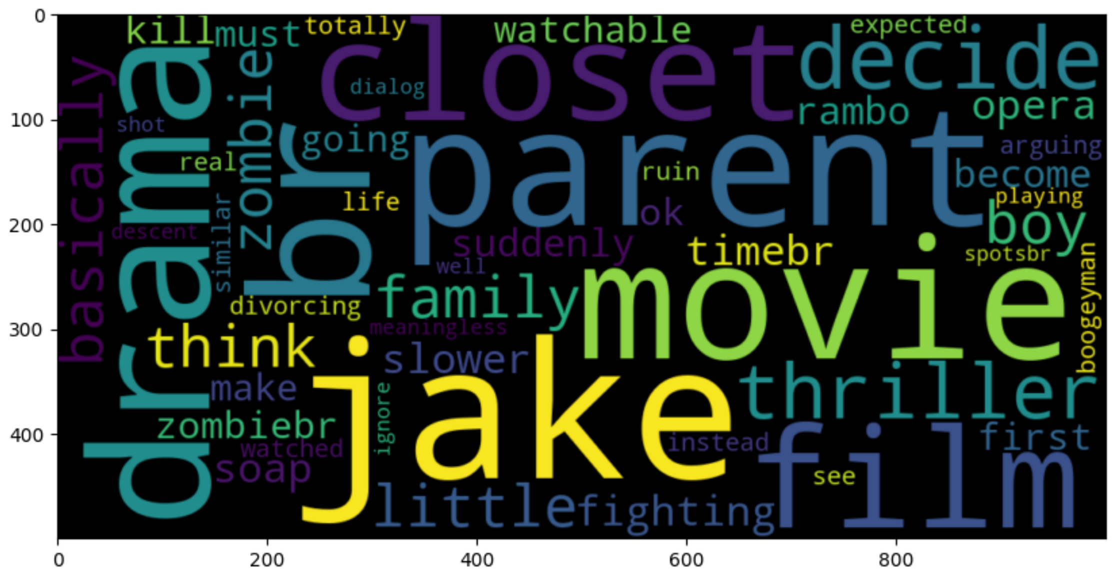
WordCloud for Negative Words.

[31]:

```
plt.figure(figsize=(10,10))
text=df["review"][3]
WC=WordCloud(width=1000,height=500,max_words=500,min_font_size=5)
words=WC.generate(text)
plt.imshow(words,interpolation='bilinear')
plt.show
```

[31]:

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Saving The Model

[62]:

```
pickle.dump(model, open('rf.pkl', 'wb'))
```

Conclusion :

- We have observed that ensemble techniques work better on NLP projects compared to Naive Bayes or SVM.
- Model accuracy can further be improved by using lexicon models like Textblob.
- Such analysis are required by the companies/businesses to gain insights about how customers feel about certain topics, and detect urgent issues in real time before they spiral out of control.