Activity 6

Sentiment Analysis on the processed IMDB Movie Reviews and Ratings

In [1]:

```
import csv
from bs4 import BeautifulSoup
import requests
import pandas as pd
import time
time.sleep(2)
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
urls = []
url1 = 'https://www.imdb.com/title/tt0082971/reviews?ref_=tt_urv'
url2 = 'https://www.imdb.com/title/tt0401997/reviews?ref_=tt_urv'
url3 = 'https://www.imdb.com/title/tt9686790/reviews?ref_=tt_urv'
url4 = 'https://www.imdb.com/title/tt8108198/reviews?ref_=tt_urv'
url5 = 'https://www.imdb.com/title/tt0072684/reviews?ref_=tt_urv'
url6 = 'https://www.imdb.com/title/tt7131622/reviews?ref_=tt_urv'
url7 = 'https://www.imdb.com/title/tt22488728/reviews?ref_=tt_urv'
```

In [3]:

```
urls.append(url1)
urls.append(url2)
urls.append(url3)
urls.append(url4)
urls.append(url5)
urls.append(url6)
urls.append(url7)
```

In [4]:

```
content = []
for url in urls:
 page = requests.get(url, timeout=2.50)
 page_content = page.content
 soup = BeautifulSoup(page_content, 'html.parser')
 content.append(soup.find_all('div', class_= 'review-container'))
```

In [5]:

```
print(content)
```

```
[[<div class="review-container">
<div class="lister-item-content">
<div class="ipl-ratings-bar">
<span class="rating-other-user-rating">
<svg class="ipl-icon ipl-star-icon" fill="#000000" height="24" viewbox
="0 0 24 24" width="24" xmlns="http://www.w3.org/2000/svg">
<path d="M0 0h24v24H0z" fill="none"></path>
<path d="M12 17.27L18.18 21l-1.64-7.03L22 9.24l-7.19-.61L12 2 9.19 8.63
2 9.24l5.46 4.73L5.82 21z"></path>
<path d="M0 0h24v24H0z" fill="none"></path>
</svg>
<span>9</span><span class="point-scale">/10</span>
</span>
</div>
<a class="title" href="/review/rw8212272/"> Perpetually Entertaining...
</a> <div class="display-name-date">
<span class="display-name-link"><a href="/user/ur4103165/">Xstal</a></s
pan><span class="review-date">6 June 2022</span>
</div>
```

In [6]:

```
movie = pd.DataFrame(columns=['Review','Rating'])
```

In [7]:

```
review = []
rating = []
count = 0
for cc in content:
    for c in cc:
        count+= 1

        print('\nMovie review ', count)
        #Get review.
        str = c.find_all('a', attrs={'class':'title'})
        rReview =''
        for s in str:
            #print('Review is: ',s.get_text())
            rReview = s.get_text()

            #Get rating.
            ratings = c.find_all('span', attrs={'class':''})
            rVal = []
            for r in ratings:
                str1 = r.get_text().strip()
                rVal.append(str1)

                val = rVal[0]

                if(len(val) > 2):
                   continue
                else:
                    review.append(rReview)
                    rating.append(val)

                    print('Review: ', rReview)
                    print('Rating: ',val)

movie['Review'] = review
movie['Rating'] = rating
```

```
Movie review  1
Review:   Perpetually Entertaining...

Rating:  9
Review:   Perpetually Entertaining...

Rating:  9

Movie review  2
Review:   When Harrison Ford was crapping gold

Rating:  9
Review:   When Harrison Ford was crapping gold

Rating:  9

Movie review  3
Review:   There will never be another film like Raiders
```

In [8]:

```
movie.head()
```

Out[8]:

	Review	Rating
0	Perpetually Entertaining...\n	9
1	Perpetually Entertaining...\n	9
2	When Harrison Ford was crapping gold\n	9
3	When Harrison Ford was crapping gold\n	9
4	There will never be another film like Raiders\n	10

In [9]:

```
movie.shape
```

Out[9]:

```
(326, 2)
```

In [10]:

```
movie.to_csv('AbidemiAleem-2128712-Review.csv', index=False)
```

In [11]:

```
import string
import re
#import nltk
#nltk.download()
from nltk.corpus import stopwords
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.model_selection import train_test_split
```

In [12]:

```
textFeatures = movie['Review'].copy()
textFeatures.shape
```

Out[12]:

```
(326,)
```

In [13]:

```
#Preparing text for Wordcloud
text = []
for t in textFeatures:
 text.append(t)
all_text = ', '.join(t for t in text)
#print(all_text)
print(len(all_text))
```

```
13008
```

In [14]:

```
from os import path
from PIL import Image
from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [15]:

```
# Create stopword list
stopwords = set(STOPWORDS)
stopwords.update(["br", "im", "thats"]) #"im","lol","Xa","film"])
# Generate a word cloud image
wordcloud = WordCloud(stopwords=stopwords, background_color="white").generate(all_text)
# Display the image
plt.imshow(wordcloud, interpolation='bilinear')
plt.axis("off")
plt.show()
#save the generated image to a file
#wordcloud.to_file("wordcloud_cb_all.png")
```

In [16]:

```
import nltk
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     C:\Users\bidam\AppData\Roaming\nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

In [17]:

```
sid = SentimentIntensityAnalyzer()
c = 0
for t in text:
    c+=1
    print(c, t)
    ss = sid.polarity_scores(t)
    print(ss)

    if(ss['compound'] >= 0.05):
        print('positive')

    elif(ss['compound'] <= -0.05):
        print('negative')

    else:
        print('neutral')
    print('\n')
```

```
1  Perpetually Entertaining...

{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
neutral


2  Perpetually Entertaining...

{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
neutral


3  When Harrison Ford was crapping gold

{'neg': 0.0, 'neu': 1.0, 'pos': 0.0, 'compound': 0.0}
neutral


4  When Harrison Ford was crapping gold
```

In [18]:

```
label = []

for r in movie['Rating']:
    r = int(r)
    if (r>5):
        label.append('1') #Positive
    elif(r<5):
        label.append('-1') #Negative
    elif(r==5):
        label.append('0') #Netural

movie['class-label'] = label
```

In [19]:

```
movie['class-label'].value_counts()
```

Out[19]:

```
1     256
-1     50
0      20
Name: class-label, dtype: int64
```

In [20]:

```
movie = movie[movie['class-label']!='0']
```

In [21]:

```
movie['class-label'].value_counts()
```

Out[21]:

```
1     256
-1     50
Name: class-label, dtype: int64
```

In [22]:

```
textFeatures = movie['Review'].copy()
textFeatures.shape
```

Out[22]:

```
(306,)
```

In [23]:

```
import nltk
nltk.download('punkt')
# Stemming using TextBlob library for stemming
from textblob import TextBlob
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\bidam\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

In [24]:

```
def textblob_tokenizer(input_str):
    blob = TextBlob(input_str.lower())
    tokens = blob.words
    words = [token.stem() for token in tokens]
    return words
```

In [25]:

```
#Toy example:
print(textblob_tokenizer('Q: studed studing!!! I miss uuuu! It&#039;s'))
```

```
['q', 'stude', 'stude', 'i', 'miss', 'uuuu', 'it', '039', 's']
```

In [26]:

```
# Try it for fun:
def textblob_tokenizer(text):
    blob = TextBlob(text)
    tokens = [token for token in blob.words if token.isalpha()]
    return tokens

text = 'Q: studed studing!!! I miss uuuu! It&#039;s'
token_list = textblob_tokenizer(text)
print(token_list)
print(len(token_list))
```

```
['Q', 'studed', 'studing', 'I', 'miss', 'uuuu', 'It', 's']
8
```

Observation: The length of the list is 8 once the non-alphabetic characters are removed.

In [27]:

```
#countvectorizer convers each review into a vector based on the word count.
countvectorizer = CountVectorizer(analyzer= 'word', stop_words= 'english',
                                  tokenizer=textblob_tokenizer)

#convers text into a vector based on tf-idf weighting scheme.
tfidfvectorizer = TfidfVectorizer(analyzer= 'word', stop_words= 'english',
                                  tokenizer=textblob_tokenizer)
```

In [28]:

```
textFeatures
```

Out[28]:

```
0                          Perpetually Entertaining...\n
1                          Perpetually Entertaining...\n
2                 When Harrison Ford was crapping gold\n
3                 When Harrison Ford was crapping gold\n
4       There will never be another film like Raiders\n
                              ...
321                           One Good film for Adults\n
322                            A thorough entertainer!\n
323                            A thorough entertainer!\n
324                      Hilarious - Heart Warming ❤❤❤\n
325                      Hilarious - Heart Warming ❤❤❤\n
Name: Review, Length: 306, dtype: object
```

In [29]:

```
count_matrix = countvectorizer.fit_transform(textFeatures)
tfidf_matrix = tfidfvectorizer.fit_transform(textFeatures)
```

In [30]:

```
print(tfidf_matrix) #print elements of the matrix.
```

```
  (0, 104)      0.6727326859628675
  (0, 239)      0.7398856217262137
  (1, 104)      0.6727326859628675
  (1, 239)      0.7398856217262137
  (2, 140)      0.5231781404250305
  (2, 68)       0.5231781404250305
  (2, 127)      0.47569384417018373
  (2, 150)      0.47569384417018373
  (3, 140)      0.5231781404250305
  (3, 68)       0.5231781404250305
  (3, 127)      0.47569384417018373
  (3, 150)      0.47569384417018373
  (4, 260)      0.7188309874897008
  (4, 188)      0.5315523047041081
  (4, 120)      0.44803365809761725
  (5, 260)      0.7188309874897008
  (5, 188)      0.5315523047041081
  (5, 120)      0.44803365809761725
  (6, 5)        0.4469928548721696
  (6, 3)        0.4677134258593081
  (6, 326)      0.5811523804723088
  (6, 51)       0.4936632958156257
  (7, 5)        0.4469928548721696
  (7, 3)        0.4677134258593081
  (7, 326)      0.5811523804723088
  :     :
  (295, 50)     0.44032912705259564
  (296, 95)     0.6659610718364615
  (296, 102)    0.6055176198153439
  (296, 219)    0.43571121501700194
  (297, 95)     0.6659610718364615
  (297, 102)    0.6055176198153439
  (297, 219)    0.43571121501700194
  (298, 344)    1.0
  (299, 344)    1.0
  (300, 4)      0.731845686115355
  (300, 142)    0.5062935910079606
  (300, 120)    0.4561454717725577
  (301, 4)      0.731845686115355
  (301, 142)    0.5062935910079606
  (301, 120)    0.4561454717725577
  (302, 322)    0.7621439272097792
  (302, 103)    0.6474076260110432
  (303, 322)    0.7621439272097792
  (303, 103)    0.6474076260110432
  (304, 342)    0.606163561298404
  (304, 151)    0.606163561298404
  (304, 153)    0.5149091899627273
  (305, 342)    0.606163561298404
  (305, 151)    0.606163561298404
  (305, 153)    0.5149091899627273
```

In [31]:

```
print(tfidf_matrix.shape)
print(count_matrix.shape)
```

```
(306, 357)
(306, 357)
```

In [32]:

```
features_train, features_test, labels_train, labels_test = train_test_split(
 tfidf_matrix, movie['class-label'], test_size=0.3,random_state=8)
print(features_train.shape, features_test.shape, labels_train.shape, labels_test.shape)
```

```
(214, 357) (92, 357) (214,) (92,)
```

In [33]:

```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import accuracy_score
```

In [34]:

```
#SVM classifier
from sklearn.svm import SVC
print("\nEvaluation for SVM \n")
svc = SVC(kernel='sigmoid', gamma=1.0)
svc.fit(features_train, labels_train)
prediction = svc.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy:', acc)

from sklearn.metrics import precision_score
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision:', prec)

from sklearn.metrics import recall_score
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall:', recall)

from sklearn.metrics import f1_score
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ', f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
#print(prediction)
```

```
Evaluation for SVM

Accuracy: 0.8260869565217391
Precision: 0.8570048309178743
Recall: 0.8260869565217391
F-1 measure:  0.7650053022269353

Confusion Matrix:

[[ 2 16]
 [ 0 74]]
              precision    recall  f1-score   support

          -1       1.00      0.11      0.20        18
           1       0.82      1.00      0.90        74

    accuracy                           0.83        92
   macro avg       0.91      0.56      0.55        92
weighted avg       0.86      0.83      0.77        92
```

In [35]:

```
#Decision Tree
print("\nEvaluation for Decision Tree \n")
from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()
dtree.fit(features_train, labels_train)
prediction = dtree.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy: ', acc)
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision: ', prec)
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall: ', recall)
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ',f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
```

```
Evaluation for Decision Tree

Accuracy:  0.8913043478260869
Precision:  0.9042443064182194
Recall:  0.8913043478260869
F-1 measure:  0.8738410736209304

Confusion Matrix:

[[ 8 10]
 [ 0 74]]
              precision    recall  f1-score   support

          -1       1.00      0.44      0.62        18
           1       0.88      1.00      0.94        74

    accuracy                           0.89        92
   macro avg       0.94      0.72      0.78        92
weighted avg       0.90      0.89      0.87        92
```

In [36]:

```
# load the reviews data from a CSV file
df = pd.read_csv('AbidemiAleem-2128712-Review.csv')

# define the positive and negative reviews
positive_reviews = df[df['Rating'] >= 6]
negative_reviews = df[df['Rating'] <= 4]
neutral_reviews = df[df['Rating'] == 5]

# generate the wordcloud for positive reviews
positive_text = ' '.join(positive_reviews['Review'])
positive_wordcloud = WordCloud(background_color='white').generate(positive_text)
plt.imshow(positive_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()

# generate the wordcloud for negative reviews
negative_text = ' '.join(negative_reviews['Review'])
negative_wordcloud = WordCloud(background_color='white').generate(negative_text)
plt.imshow(negative_wordcloud, interpolation='bilinear')
plt.axis('off')
plt.show()
```

Try-It-Yourself A:

Based on the ratings, the worldclouds show the most frequently used terms in both positive and negative reviews separately. Both use the word "Film" and "Movie" most frequently. We can conclude that the two worldclouds are easily distinguishable. While positive ratings demonstrate viewers' happiness and negative reviews demonstrate viewers' dissatisfaction, it offers insights into what consumers value or dislike.

In [37]:

```
# Try-It-Yourself B:
# Create a CountVectorizer object to tokenize the text data and create the BoW matrix
bow_vectorizer = CountVectorizer(stop_words='english')
bow_matrix = bow_vectorizer.fit_transform(movie['Review'])

# Split the dataset into training and testing sets
bow_train, bow_test, labels_train, labels_test = train_test_split(
    bow_matrix, movie['class-label'], test_size=0.3, random_state=8)

# Print the dimensions of the training and testing sets
print("BoW training set shape:", bow_train.shape)
print("BoW testing set shape:", bow_test.shape)
print("Labels training set shape:", labels_train.shape)
print("Labels testing set shape:", labels_test.shape)
```

```
BoW training set shape: (214, 375)
BoW testing set shape: (92, 375)
Labels training set shape: (214,)
Labels testing set shape: (92,)
```

In [38]:

```
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(count_matrix, movie['class-label'],
                                                    test_size=0.3, random_state=42)


# Train an SVM classifier
from sklearn.svm import SVC
svc = SVC(kernel='linear', C=1, gamma='scale')
svc.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = svc.predict(X_test)

# Evaluate the performance of the classifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report
print("Evaluation for SVM using BoW scheme:\n")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred, average='weighted'))
print("Recall:", recall_score(y_test, y_pred, average='weighted'))
print("F1-score:", f1_score(y_test, y_pred, average='weighted'))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Evaluation for SVM using BoW scheme:

Accuracy: 0.9347826086956522
Precision: 0.9396739130434782
Recall: 0.9347826086956522
F1-score: 0.9295313382269904
Confusion Matrix:
 [[12  6]
 [ 0 74]]
Classification Report:
              precision    recall  f1-score   support

          -1       1.00      0.67      0.80        18
           1       0.93      1.00      0.96        74

    accuracy                           0.93        92
   macro avg       0.96      0.83      0.88        92
weighted avg       0.94      0.93      0.93        92
```

In [40]:

```
#Decision Tree
print("\nEvaluation for Decision Tree \n")
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier()
dtree.fit(features_train, labels_train)
prediction = dtree.predict(features_test)
acc = accuracy_score(labels_test,prediction)
print('Accuracy: ', acc)
prec = precision_score(labels_test,prediction, average='weighted')
print('Precision: ', prec)
recall = recall_score(labels_test,prediction, average='weighted')
print('Recall: ', recall)
f1 = f1_score(labels_test,prediction, average='weighted')
print('F-1 measure: ',f1)
print('\nConfusion Matrix:\n')
print(confusion_matrix(labels_test, prediction))
print(classification_report(labels_test, prediction))
```

```
Evaluation for Decision Tree

Accuracy:  0.8913043478260869
Precision:  0.9042443064182194
Recall:  0.8913043478260869
F-1 measure:  0.8738410736209304

Confusion Matrix:

[[ 8 10]
 [ 0 74]]
              precision    recall  f1-score   support

          -1       1.00      0.44      0.62        18
           1       0.88      1.00      0.94        74

    accuracy                           0.89        92
   macro avg       0.94      0.72      0.78        92
weighted avg       0.90      0.89      0.87        92
```

Report:

Using different feature sets, an SVM classifier is trained to generate the two outputs. The TF-IDF (Term Frequency-Inverse Document Frequency) representation of the input text is used to produce the first result, while the BoW (Bag of Words) technique is used to produce the second.

For the evaluation (Evaluation for SVM) using TF-IDF, the accuracy is 0.826, precision is 0.857, recall is 0.826, and F1-score is 0.765. The evaluation (Evaluation for SVM using BoW scheme) has an accuracy of 0.935, precision of 0.940, recall of 0.935, and F1-score of 0.930. The confusion matrices for both evaluations also show that the BoW scheme performs better at correctly classifying instances in both classes.

Comparing the two models' decision trees Results:

In terms of accuracy, precision, recall, and F-1 measure, as well as the confusion matrix and classification report, the outcomes for the evaluation of the Decision Tree utilising TF-IDF and BOW are equal.

REFRENCES:

[1] Dr Vinita Nahar-Workshop 6 - Sentiment Analysis on IMDB Movie Review notebook

[2] GeeksforGeeks. (2021). Understanding TF-IDF (Term Frequency-Inverse Document Frequency). [online] Available at: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/).

[3] GeeksforGeeks. (2019). Bag of words (BoW) model in NLP. [online] Available at: https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/ (https://www.geeksforgeeks.org/bag-of-words-bow-model-in-nlp/).

[4] Mujtaba, H. (2020). An introduction to Bag of Words in NLP using Python. [online] GreatLearning. Available at: https://www.mygreatlearning.com/blog/bag-of-words/ (https://www.mygreatlearning.com/blog/bag-of-words/).

[5] Shivanandhan, M. (2020). What is Sentiment Analysis? A Complete Guide for Beginners. [online] freeCodeCamp.org. Available at: https://www.freecodecamp.org/news/what-is-sentiment-analysis-a-complete-guide-to-for-beginners/ (https://www.freecodecamp.org/news/what-is-sentiment-analysis-a-complete-guide-to-for-beginners/).

In [ ]: