NAME :RETHINAGIRI G

ROLL NO : 225229130

COURSE TITLE : NATURAL LANGUAGE PRE-PROCESSING LAB

LAB.07 Sentiment Analysis on Movie Reviews

Exercise -I

1.Open the file

In [1]:
```
import pandas as pan
```

In [2]:
```
train=pan.read_csv("train.tsv",sep='\t')
```

2. Print Basic Statistics

In [3]:
```
train.head()
```

Out[3]:

	PhraseId	SentenceId	Phrase	Sentiment
0	1	1	A series of escapades demonstrating the adage ...	1
1	2	1	A series of escapades demonstrating the adage ...	2
2	3	1	A series	2
3	4	1	A	2
4	5	1	series	2

In [4]:
```
train.shape
```

Out[4]:
```
(156060, 4)
```

In [5]:
```
train.describe()
```

Out[5]:

	PhraseId	SentenceId	Sentiment
count	156060.000000	156060.000000	156060.000000
mean	78030.500000	4079.732744	2.063578
std	45050.785842	2502.764394	0.893832
min	1.000000	1.000000	0.000000
25%	39015.750000	1861.750000	2.000000
50%	78030.500000	4017.000000	2.000000
75%	117045.250000	6244.000000	3.000000
max	156060.000000	8544.000000	4.000000

In [6]:
```
train.columns
```

Out[6]:
```
Index(['PhraseId', 'SentenceId', 'Phrase', 'Sentiment'], dtype='object')
```

3.How many reviews exist for each sentiment?

In [7]:
```
train['Sentiment'].value_counts()
```

Out[7]:
```
2    79582
3    32927
1    27273
4     9206
0     7072
Name: Sentiment, dtype: int64
```

Exercise-II

1. Extract 200 reviews for each sentiment and store them in small_rotten_train.csv file

In [8]:
```
a = train.loc[train.Sentiment==0]
b = train.loc[train.Sentiment==1]
c = train.loc[train.Sentiment==2]
d = train.loc[train.Sentiment==3]
e = train.loc[train.Sentiment==4]
```

In [9]:
```
small_rotten_train = pan.concat([a[0:200],b[0:200],c[0:200],d[0:200],e[0:200]])
```

In [10]:
```
small_rotten_train.head()
```

Out[10]:

	PhraseId	SentenceId	Phrase	Sentiment
101	102	3	would have a hard time sitting through this one	0
103	104	3	have a hard time sitting through this one	0
157	158	5	Aggressive self-glorification and a manipulati...	0
159	160	5	self-glorification and a manipulative whitewash	0
201	202	7	Trouble Every Day is a plodding mess .	0

In [11]:
```
small_rotten_train.to_csv("small_rotten_train.csv")
```

EXERCISE-3

1. Open the file 'small_rotten_train.csv'

In [12]:
```
file=pan.read_csv('small_rotten_train.csv')
```

In [13]:
```
file.head()
```

Out[13]:

	Unnamed: 0	PhraseId	SentenceId	Phrase	Sentiment
0	101	102	3	would have a hard time sitting through this one	0
1	103	104	3	have a hard time sitting through this one	0
2	157	158	5	Aggressive self-glorification and a manipulati...	0
3	159	160	5	self-glorification and a manipulative whitewash	0
4	201	202	7	Trouble Every Day is a plodding mess .	0

2. The review text are stored in "Phrase" column. Extract that into a separate DataFrame, say "X"

In [14]:
```
X=file["Phrase"]
X.head()
```

Out[14]:
```
0      would have a hard time sitting through this one
1            have a hard time sitting through this one
2    Aggressive self-glorification and a manipulati...
3      self-glorification and a manipulative whitewash
4                Trouble Every Day is a plodding mess .
Name: Phrase, dtype: object
```

3. The 'Sentiment' column is your target, say '"y'

In [15]:
```
y=file["Sentiment"]
y.head()
```

Out[15]:
```
0    0
1    0
2    0
3    0
4    0
Name: Sentiment, dtype: int64
```

4. Perform pre-processing: convert into lower case, remove stop words and lemmatize. The following function will help you.

In [16]:
```
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

import warnings
warnings.filterwarnings('ignore')

import nltk
nltk.download('wordnet')

from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()

import nltk
nltk.download('omw-1.4')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     C:\Users\user\AppData\Roaming\nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[16]:
```
True
```

In [ ]:
```
```

In [17]:
```
def clean_review(review):
    tokens = review.lower().split()
    filtered_tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop_words]
    return " ".join(filtered_tokens)
```

5. Apply the above function to X

In [18]:
```
X_clean = X.apply(lambda x: clean_review(x))
```

6. Split X and y for training and testing (Use 20% for testing)

In [19]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X_clean,y,train_size = 0.8,test_size=0.2,random_state=42)
```

7. Create TfidfVectorizer as below and perform vectorization on X_train using fit_perform() method.

In [20]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer = TfidfVectorizer(min_df = 3,max_features=None,ngram_range=(1,2),use_idf=1)
```

In [21]:
```
X_train_tf = vectorizer.fit_transform(X_train)
X_test_tf = vectorizer.transform(X_test)
```

8. Create MultinomialNB model and perform training using X_train_lemmatized and y_train

In [22]:
```
from sklearn.naive_bayes import MultinomialNB
clf = MultinomialNB()

clf.fit(X_train_tf,y_train)
```

Out[22]:
```
MultinomialNB()
```

9. Perform validation on X_test and predict output

In [23]:
```
y1_pred = clf.predict(X_test_tf)

y1_pred
```

Out[23]:
```
array([2, 1, 1, 3, 2, 4, 2, 4, 4, 0, 4, 0, 1, 4, 4, 4, 1, 4, 3, 1, 3, 3,
       0, 2, 1, 2, 0, 4, 0, 0, 1, 0, 3, 2, 0, 0, 1, 1, 0, 0, 1, 3, 1, 4,
       3, 1, 0, 3, 2, 1, 2, 4, 1, 2, 1, 0, 2, 2, 0, 4, 0, 0, 0, 1, 3, 3,
       3, 4, 0, 2, 0, 2, 0, 1, 1, 3, 3, 0, 0, 1, 4, 0, 1, 0, 1, 2, 4, 3,
       3, 1, 1, 2, 2, 2, 3, 0, 3, 0, 0, 0, 3, 1, 4, 1, 2, 3, 0, 4, 0, 4,
       2, 3, 4, 4, 0, 2, 0, 0, 2, 1, 2, 2, 0, 0, 2, 2, 1, 2, 4, 3, 2, 3,
       1, 4, 2, 2, 0, 2, 1, 4, 0, 1, 3, 0, 2, 4, 4, 2, 3, 1, 4, 0, 0, 2,
       4, 2, 1, 3, 0, 4, 2, 3, 0, 3, 4, 3, 3, 3, 0, 1, 3, 2, 0, 4, 0, 1,
       1, 0, 1, 0, 2, 3, 1, 2, 4, 0, 1, 1, 4, 3, 4, 4, 3, 0, 2, 2, 1, 1,
       3, 0], dtype=int64)
```

10. Print classification _report and accuracy_score

In [24]:
```
from sklearn.metrics import accuracy_score, classification_report

print("Accuracy Score is =", accuracy_score(y_test, y1_pred))

print("Classification Report : \t",classification_report(y_test, y1_pred))
```

```
Accuracy Score is = 0.68
Classification Report :                       precision    recall  f1-score   support

           0       0.52      0.79      0.63        33
           1       0.73      0.62      0.67        48
           2       0.68      0.73      0.70        37
           3       0.69      0.63      0.66        38
           4       0.85      0.66      0.74        44

    accuracy                           0.68       200
   macro avg       0.69      0.69      0.68       200
weighted avg       0.70      0.68      0.68       200
```

Exercise-IV

1.Open test.tsv file into dataframe

In [25]:
```
test = pan.read_csv("test.tsv", sep='\t')

test.head()
```

Out[25]:

	PhraseId	SentenceId	Phrase
0	156061	8545	An intermittently pleasing but mostly routine ...
1	156062	8545	An intermittently pleasing but mostly routine ...
2	156063	8545	An
3	156064	8545	intermittently pleasing but mostly routine effort
4	156065	8545	intermittently pleasing but mostly routine

In [26]:
```
X1 = test['Phrase']
```

2. Clean the test data using the clean_review function

In [27]:
```
X1_clean = X1.apply(lambda x: clean_review(x))
```

3. Build TFIDF values using transform() method

In [28]:
```
X1_train_tf = vectorizer.transform(X1_clean)
```

4. Perform prediction using predict() method

In [29]:
```
y1_pred = clf.predict(X1_train_tf)

y1_pred
```

Out[29]:
```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```