CSCI 544 - Homework Assignment No 1

`Version` : 1.0

`Editor` : Shubham Sanjay Darekar

`Date` : 9/11/2023 `Execution Time` : ~ 57 min

Importing the required libraries

In [ ]:
```
import pandas as pd
import numpy as np
import nltk
import re
import contractions
from bs4 import BeautifulSoup
from sklearn.metrics import precision_recall_fscore_support
from itertools import chain
from nltk.corpus import stopwords

## Approx run time - 2s
```

Breif description of usage of all the libraries imported

1. Pandas - Used to read and manupulate the data using dataframe
2. NumPy - Used to manupulate the numeric values in the dataset
3. NLTK - Natural Language toolkit - functions such as stopwords removal, lemmatization, etc are done using this library
4. Re - Regular Expression library used to substitute characters while removing stop words
5. Contractions - Used to expand the contractions
6. BS4 - Used to remove the HTML and XML data
7. Sklearn - This module has implementation of all the ML models used in the solution
8. itertools - Used to flatten out the array

Downloading the required corpus/libraries from NLTK

In [ ]:
```
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')

## Approx run time - 1s
```

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

Out[ ]: True

In [ ]:

```
# to not print the warnings
import warnings
warnings.filterwarnings('ignore')
```

Read Data

To read the data read_csv method from pandas is used.
`Parameters` :

1. sep ~ Seperater - \t as the values are tab seperated
2. engine ~ Using python engine to avoid unsupported format by C engine - `Ref` - https://stackoverflow.com/questions/52774459/engines-in-python-pandas-read-csv
3. quoting ~ Set to 3 i.e none to control the quoting field behaviour - `Ref` - https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html

In [ ]:

```
reviews_ratings = pd.read_csv("D:\\Applied NLP\\HW1\\amazon_reviews_us_Office_Produ

## Approx run time - 1 min 10 s
```

Keep Reviews and Ratings

As the review headline and review body are the fields using in classifier and star rating is the field used to label the categories just slicing those fields. The review headline and review body are concatenated with a space in between as headline is helping in model performance.

In [ ]:

```
reviews_ratings = reviews_ratings[['review_headline','review_body','star_rating']]
reviews_ratings['review_headline_body'] = reviews_ratings['review_headline'] + " "

## Approx run time - 4s
```

We form two classes and select 50000 reviews randomly from each class.

1. The first class contains the reviews with ratings less than or equal to 3. We are selecting random 50k reviews.
2. The second class contains the reviews with ratings greater than 3. We are selecting random 50k reviews.

In [ ]:
```
#Random selection of 50,000 rows from each class
df_class1 = reviews_ratings[reviews_ratings['star_rating']>3].sample(n=50000, rando
df_class1['Class'] = 2
df_class2 = reviews_ratings[reviews_ratings['star_rating']<=3].sample(n=50000, rand
df_class2['Class'] = 1

final_dataset = pd.concat([df_class1, df_class2], ignore_index=True, sort=False).re

## Approx run time - 1s
```

As the following variables are not going to be used in future, deleting the variable references.

In [ ]:
```
#deleting to reduce memory usage
del df_class1
del df_class2
del final_dataset['review_headline']
del final_dataset['review_body']
del reviews_ratings

## Approx run time - 1s
```

Data Cleaning

For data cleaning following steps are performed

> 1. Using the `bs4/ Beautiful Soup` library we are just keeping the text and removing, if any, HTML or XML tags
> 2. Using the regular expression library, we are splitting the sentence on basis of spaces, commas, colon, semi-colon and periods. This is done in order to pass the values to the next step
> 3. Using the `contractions library`, we are expanding the contractions and again splitting the words
> 4. As the output of last step is list of lists, using the chain method to flatten out the array

> 5. Using the regex `'[^A-Za-z0-9\s]+'` replacing all the non Alphanumeric characters with space
> 6. Removing the extra spaces in between the words generated as result of previous output

In [ ]:
```
## Calculating length before data cleaning
len_before_data_cleaning = final_dataset['review_headline_body'].apply(str).apply(l

## Approx run time - 1s
```

In [ ]:
```
#function to clean the text
def data_clean(s:str):
    s = BeautifulSoup(str(s)).get_text() #removing HTML and XML tags
    s = re.split(" |\,|;|\.|:",s) #splitting on basis of spaces, commas, colon, sem
    s = [contractions.fix(x).lower().strip().split() for x in s] #expanding contrac
    s = list(chain(*s)) #Flattening out the array

    #Keeping the Alpha Numeric characters and spaces
    s = (" ").join([re.sub(r'[^A-Za-z0-9\s]+', ' ', word) for word in s])

    #removing extra spaces
    s = (" ").join(s.split()).strip()

    return s

## Approx run time - 1s
```

In [ ]:
```
#applying the data cleaning function
final_dataset['review_headline_body'] = final_dataset['review_headline_body'].apply

## Approx run time - 1 min
```

In [ ]:
```
## Calculating the length after the data cleaning
len_after_data_cleaning = final_dataset['review_headline_body'].apply(str).apply(le

## Approx run time - 1s
```

> - Average Length before cleaning - `339`
> - Average Length after cleaning - `326`
> - Difference = `13 characters`

In [ ]:
```
print(str(round(len_before_data_cleaning,2))+" "+str(round(len_after_data_cleaning,

## Approx run time - 1s
```

```
339.68 326.76
```

Pre-processing

Remove the stop words

In order to remove the stop words the following function is written. The function tokenizes the words and then removes the stopwords that are present in the given string from list of NLTK's stopwords corpus.

In [ ]:
```
#function to remove stop words
def remove_stop(s:str):
    stopw = set(stopwords.words("english")) #creating set of stop words from NLTK
    tokens = nltk.tokenize.word_tokenize(s) # generating tokens
    tokens = [x for x in tokens if x not in stopw] # removing the stop words
    return tokens

## Approx run time - 1s
```

In [ ]:
```
#applying the stop words removal function
final_dataset['review_headline_body'] = final_dataset['review_headline_body'].apply

## Approx run time - 1 min 30 s
```

Perform lemmatization

> Using the NLTK's word lemmatizer we are lemmatizing the words.
>
> 1. As the NLTK's lemmatizer takes 2 inputs, the word and its pos tags (a for adjective, v for verb, n for noun and r for adverbs), the first step is to find the POS tags using the pos_tag() method and using a function to convert the POS tags into one of the acceptable inputs.
> `Ref` - https://stackoverflow.com/questions/60345476/apply-nlp-wordnetlemmatizer-on-whole-sentence-show-error-with-unknown-pos , https://www.nltk.org/api/nltk.stem.wordnet.html?highlight=wordnetlemmatizer
> 2. This function returns the final string of lemmatized words from the given input

In [ ]:
```
from nltk.stem import WordNetLemmatizer
wnl = WordNetLemmatizer()

## ref: https://stackoverflow.com/questions/60345476/apply-nlp-wordnetlemmatizer-on
def switch_pos(tag):
    if tag.startswith('J'):
        return 'a'
    elif tag.startswith('V'):
        return 'v'
    elif tag.startswith('N'):
        return 'n'
    return 'r'
```

```python
def custom_lemmatize(s:list):
    pos_tagged = nltk.pos_tag(s) #pos tagging with nltk
    #lemmatizing by using the pos tags
    tokens = [wnl.lemmatize(x[0],switch_pos(x[1])) for x in pos_tagged]
    return " ".join(tokens) #returning in sentence format

## Approx run time - 1s
```

In [ ]:
```python
#applying the custom lemmitization function on dataset
final_dataset['review_headline_body'] = final_dataset['review_headline_body'].apply

## Approx run time - 5 min 45 s
```

In [ ]:
```python
#calculating average length of the sentence after preprocessing
len_after_data_preprocessing = final_dataset['review_headline_body'].apply(len).mea

## Approx run time - 1s
```

> - Average Length before preprocessing - `326`
> - Average Length after preprocessing - `198`
> - Difference = `128 characters`

In [ ]:
```python
print(str(round(len_after_data_cleaning,2))+" "+str(round(len_after_data_preprocess

## Approx run time - 1s
```

```
326.76 198.26
```

TF-IDF and BoW Feature Extraction

Using the CountVectorizer from Sci-kit learn, Bag of words feature extraction can be achieved. The data type of the output has been changed to int8 as the maximum value of a feature will be in 80s and it saves a lot of computing memory. It is observed that with ngram_range = (1,4), there is significant increase in performance. This is because the number of features are increased by using upto 4 words together as a single feature. This value was change to (1,3), (1,2) and (1,5) but (1,4) performed better in all the models.

In [ ]:
```python
# Bag of words feature extraction
from sklearn.feature_extraction.text import CountVectorizer
"""
    1. reducing the size to int8 to avoid memory issues - dtype = int8
    2. using ngram_range to consider 1 to 4 words together while extracting the fea
"""
bow = CountVectorizer(dtype = np.int8,ngram_range=(1,4))
bow_vectors = bow.fit_transform(final_dataset['review_headline_body'])

## Approx run time - 1 min
```

Using the TfidfVectorizer from Sci-kit learn, Bag of words feature extraction can be achieved. It is observed that with ngram_range = (1,4), there is significant increase in performance. This is because the number of features are increased by using upto 4 words together as a single feature. This value was change to (1,3), (1,2) and (1,5) but (1,4) performed better in all the models.
min_df is set to 2 in order to set the minimum frequency of words to 2 to avoid the unuseful features.

In [ ]:

```
# TF-IDF feature extraction
from sklearn.feature_extraction.text import TfidfVectorizer
"""
    1. reducing the size to float 32 to avoid memory issues - dtype = float32
    2. using ngram_range to consider 1 to 4 words together while extracting the fea
"""
tfidf = TfidfVectorizer(dtype = np.float32, min_df=2,ngram_range=(1,4))
tfidf_vectors = tfidf.fit_transform(final_dataset['review_headline_body'])

## Approx run time - 30s
```

Train test split

Splitting the dataset- 80% for training and 20% for testing

In [ ]:

```
# Splitting the dataset- 80% for training and 20% for testing
from sklearn.model_selection import train_test_split

bow_x_train, bow_x_test, bow_y_train, bow_y_test = train_test_split(bow_vectors, fi
tfidf_x_train, tfidf_x_test, tfidf_y_train, tfidf_y_test = train_test_split(tfidf_v

## Approx run time - 1s
```

In [ ]:

```
## Function to print the performance
def eval(actual,predicted):
    prf = precision_recall_fscore_support(actual,predicted, average='binary')
    print(str(round(prf[0],4))+" "+str(round(prf[1],4))+" "+str(round(prf[2],4)))

## Approx run time - 1s
```

Perceptron Using Both Features

Perceptron model trained using bag of words features

In [ ]:
```
## Bow Perceptron
from sklearn.linear_model import Perceptron

per_bow = Perceptron(random_state=13) # Setting the random state to reproduce the r
per_bow.fit(bow_x_train,bow_y_train.values)

## Approx run time - 3s
```

Out[ ]:
```
Perceptron
Perceptron(random_state=13)
```

In [ ]:
```
bow_y_predict = per_bow.predict(bow_x_test)
eval(bow_y_test, bow_y_predict)

## Approx run time - 1s
```

```
0.891 0.8908 0.8909
```

Bag of words - Perceptron

Precision	Recall	F1 Score
0.891	0.8908	0.8909

Perceptron model trained using TF-IDF features

In [ ]:
```
## TF_IDF Perceptron
from sklearn.linear_model import Perceptron

per_tfidf = Perceptron(random_state=101) # Setting the random state to reproduce th
per_tfidf.fit(tfidf_x_train,tfidf_y_train.values)

## Approx run time - 1s
```

Out[ ]:
```
Perceptron
Perceptron(random_state=101)
```

In [ ]:
```
tfidf_y_predict = per_tfidf.predict(tfidf_x_test)
eval(tfidf_y_test,tfidf_y_predict)

## Approx run time - 1s
```

```
0.8735 0.9007 0.8869
```

TF-IDF - Perceptron

Precision	Recall	F1 Score
0.8735	0.9007	0.8869

SVM Using Both Features

SVM trained using Bag of words features

In [ ]:
```
from sklearn import svm

## Approx run time - 1s
```

In [ ]:
```
## Bow Support Vector machine
svm_bow = svm.SVC(kernel='linear', max_iter=10000) # setting max_iter to 10000 to a
svm_bow.fit(bow_x_train,bow_y_train.values)

## Approx run time - 25 min
```

Out[ ]:

```
SVC
SVC(kernel='linear', max_iter=10000)
```

In [ ]:
```
bow_y_predict_svm = svm_bow.predict(bow_x_test)
eval(bow_y_test,bow_y_predict_svm)

## Approx run time - 5 min
```

```
0.8839 0.8585 0.8711
```

Bag of words - Support vector machine

Precision	Recall	F1 Score
0.8839	0.8585	0.8711

SVM trained using TF-IDF features

In [ ]:
```
## TF-IDF Support Vector Machine
svm_tfidf = svm.SVC(kernel='linear', max_iter=10000)
svm_tfidf.fit(tfidf_x_train,tfidf_y_train.values)

## Approx run time - 17 min
```

Out[ ]:

```
SVC
SVC(kernel='linear', max_iter=10000)
```

In [ ]:
```
tfidf_y_predict_svm = svm_tfidf.predict(tfidf_x_test)
eval(tfidf_y_test,tfidf_y_predict_svm)

## Approx run time - 5 min
```

```
0.8984 0.8898 0.8941
```

TF - IDF - Support vector machine

Precision	Recall	F1 Score
0.8984	0.8898	0.8941

Logistic Regression Using Both Features

Logistic regression is trained using bag of words features. The max iterations are set to 1000 in order to avoid long runs

In [ ]:
```
## Bag of words Logistic Regression
from sklearn.linear_model import LogisticRegression

bow_logistic = LogisticRegression(max_iter = 1000, random_state=101) #limiting max
bow_logistic.fit(bow_x_train,bow_y_train.values)

## Approx run time - 5 min
```

Out[ ]:
```
LogisticRegression
LogisticRegression(max_iter=1000, random_state=101)
```

In [ ]:
```
bow_y_predict_logistic = bow_logistic.predict(bow_x_test)
eval(bow_y_test,bow_y_predict_logistic)

## Approx run time - 1s
```

```
0.903 0.8915 0.8972
```

Bag of words - Logistic regression

Precision	Recall	F1 Score
0.903	0.8915	0.8972

Logistic regression is trained using TF-IDF features. The max iterations are set to 5000 in order to avoid long runs

In [ ]:
```
## TF_IDF Logistic Regression
logistic_tfidf = LogisticRegression(max_iter = 5000,random_state=101) #limiting max
logistic_tfidf.fit(tfidf_x_train,tfidf_y_train.values)

## Approx run time - 20s
```

Out[ ]: ▾ LogisticRegression

```
LogisticRegression(max_iter=5000, random_state=101)
```

In [ ]:

```
tfidf_y_predict_logistic = logistic_tfidf.predict(tfidf_x_test)
eval(tfidf_y_test,tfidf_y_predict_logistic)

## Approx run time - 1s
```

```
0.8913 0.902 0.8966
```

TF-IDF - Logistic regression

Precision	Recall	F1 Score
0.8913	0.902	0.8966

Naive Bayes Using Both Features

Naive Bayes model trained using bag of words features

In [ ]:

```
## Bag of Words Naive Bayes
from sklearn.naive_bayes import MultinomialNB

bow_NB = MultinomialNB()
bow_NB.fit(bow_x_train,bow_y_train.values)

## Approx run time - 1s
```

Out[ ]: ▾ MultinomialNB

```
MultinomialNB()
```

In [ ]:

```
bow_y_predict_NB = bow_NB.predict(bow_x_test)
eval(bow_y_test,bow_y_predict_NB)

## Approx run time - 1s
```

```
0.9096 0.798 0.8501
```

Bag of words - Naive Bayes

Precision	Recall	F1 Score
0.9096	0.798	0.8501

Naive Bayes model trained using TF-IDF features

In [ ]:
```
## TD-IDF Naive Bayes
from sklearn.naive_bayes import MultinomialNB

NB_tfidf = MultinomialNB()
NB_tfidf.fit(tfidf_x_train,tfidf_y_train.values)

## Approx run time - 1s
```

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

In [ ]:
```
tfidf_y_predict_NB = NB_tfidf.predict(tfidf_x_test)
eval(tfidf_y_test,tfidf_y_predict_NB)

## Approx run time - 1s
```

```
0.8528 0.911 0.881
```

TF-IDF - Naive Bayes

Precision	Recall	F1 Score
0.8528	0.911	0.881
