CSCI 544 - Applied Natural Language Processing

CSCI 544 - Assignment 1

Name: Sri Manvith Vaddeboyina

USC ID: 1231409457

1. Data Preparation

1.1 Importing necessary libraries/packages

In [1]:

```
import re
import sys
import nltk
import pandas as pd
import numpy as np
import contractions
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
from sklearn.pipeline import Pipeline

from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.svm import LinearSVC
from sklearn.linear_model import Perceptron
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import precision_recall_fscore_support as score

nltk.download('punkt')
nltk.download('omw-1.4')
nltk.download('wordnet')
nltk.download('stopwords')

! pip install bs4
!{sys.executable} -m pip install contractions

import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package punkt to /Users/manvith/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /Users/manvith/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package wordnet to /Users/manvith/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/manvith/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!

Requirement already satisfied: bs4 in /Users/manvith/opt/miniconda3/lib/python3.9/site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in /Users/manvith/opt/miniconda3/lib/python3.9/site-packages (from
bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in /Users/manvith/opt/miniconda3/lib/python3.9/site-packages (from b
eautifulsoup4->bs4) (2.3.2.post1)
Requirement already satisfied: contractions in /Users/manvith/opt/anaconda3/lib/python3.9/site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /Users/manvith/opt/anaconda3/lib/python3.9/site-packages (fr
om contractions) (0.0.24)
Requirement already satisfied: anyascii in /Users/manvith/opt/anaconda3/lib/python3.9/site-packages (from textsea
rch>=0.0.21->contractions) (0.3.1)
Requirement already satisfied: pyahocorasick in /Users/manvith/opt/anaconda3/lib/python3.9/site-packages (from te
xtsearch>=0.0.21->contractions) (1.4.4)
```

1.2 Read Data

Reading Amazon US Beauty Reviews (tsv) dataset and retaining only the following two columns:
1. review_body
2. star_rating

In [2]:

```
df = pd.read_csv('amazon_reviews_us_Beauty_v1_00.tsv', sep='\t',usecols =['star_rating','review_body'])
```

Dropping the entire rows where any of the column contains **NA value**

In [3]:

```
df.dropna(inplace=True)
```

1.3 Keep Reviews and Ratings

Create a three-class classification problem according to the ratings.

Ratings:

1 and 2 - class 1

3 - class 2

4 and 5 - class 3

In [4]:

```
df = df[
        df['star_rating'].eq('1') |
        df['star_rating'].eq('2') |
        df['star_rating'].eq('3') |
        df['star_rating'].eq('4') |
        df['star_rating'].eq('5')
       ]
```

Verifying the datatype of each column and setting them correctly

In [5]:

```
df['star_rating']=df['star_rating'].astype(int)
```

In [6]:

```
df['review_body']=df['review_body'].astype(str)
```

Creating a 3-class classification on ratings

In [7]:

```
def condition(x):
    if x==1 or x==2:
        return 1
    elif x==3:
        return 2
    elif x==4 or x==5:
        return 3

df['rating'] = df['star_rating'].apply(condition)
```

1.4 We form three classes and select 20000 reviews randomly from each class.

Randomly selecting **20000 reviews** from each of class 1,2 and 3.

Total: 60000 reviews

In [8]:

```
df=df.groupby('rating').sample(n=20000)
```

In [9]:

```
df.drop(['star_rating'],inplace=True,axis=1)
```

Function to find the average length of reviews

In [10]:

```
def avg_len_reviews(column_name):
    length=0
    for i in range(len(df)):
        length=length+len(df[column_name].iloc[i])
    avg_length=length/len(df)
    return avg_length
```

2. Data Cleaning

Removing the following as part of data cleaning:

1. URLs

2. HTML tags

2. Contractions Expansion

3. Non-alphabetic characters

4. Converting text to lower case

5. Removing extra spaces

6. Removing emojis

In [11]:

```
def remove_urls (text):
    text = re.sub(r'(https|http)?:\/\/(\w|\.|\/|\?|\=|\&|\%)*\b', '', text, flags=re.MULTILINE)
    return(text)

def remove_contractions(text):
    expanded_words = []
    for word in text.split():
        expanded_words.append(contractions.fix(word))
    expanded_text = ' '.join(expanded_words)
    return expanded_text

def remove_emojis(data):
    emoj = re.compile("["
        u"\U0001F600-\U0001F64F"
        u"\U0001F300-\U0001F5FF"
        u"\U0001F680-\U0001F6FF"
        u"\U0001F1E0-\U0001F1FF"
        u"\U00002500-\U00002BEF"
        u"\U00002702-\U000027B0"
        u"\U00002702-\U000027B0"
        u"\U000024C2-\U0001F251"
        u"\U0001f926-\U0001f937"
        u"\U00010000-\U0010ffff"
        u"\u2640-\u2642"
        u"\u2600-\u2B55"
        u"\u200d"
        u"\u23cf"
        u"\u23e9"
        u"\u231a"
        u"\ufe0f"
        u"\u3030"
        "]+", re.UNICODE)
    return re.sub(emoj, '', data)

remove_non_english = lambda s: re.sub(r'[^a-zA-Z ]', ' ', s)
remove_spaces =  lambda s: re.sub(' +', ' ', s)
```

In [12]:

```
def cleaning(text):
        #remove urls
        text=remove_urls(text)

        #remove html tags
        text = BeautifulSoup(text, "lxml").text

        #remove contractions
        text=remove_contractions(text)

        #remove non-alphabetic chars
        text=remove_non_english(text)

        #lowercase
        text=text.lower()

        #remove extra spaces
        text=remove_spaces(text)

        #remove emojis
        text=remove_emojis(text)

        return text
```

In [13]:

```
df['cleaned_text_reviews'] = list(map(cleaning, df.review_body))
```

Average length of reviews before cleaning and after cleaning

In [14]:

```
print(avg_len_reviews("review_body"),avg_len_reviews("cleaned_text_reviews"),sep=",")
```

```
189.57826666666668,183.6413
```

3. Preprocessing

3.1 Remove the stop words

Removing the stop words in english language

Stop words are a set of commonly used words in a language. Examples of stop words in English are “a”, “the”, “is”, “are” and etc. Stop words are commonly used in Text Mining and Natural Language Processing (NLP) to eliminate words that are so commonly used that they carry very little useful information.

In [15]:

```
# remove stop words
def stop_words(text):
    text = text.split()
    stops = set(stopwords.words("english"))
    text = [w for w in text if not w in stops]
    text = " ".join(text)
    return text

df['clean_stop'] = list(map(stop_words, df.cleaned_text_reviews))
```

3.2 Perform lemmatization

Performing lemmtization - a text normalization technique used in Natural Language Processing (NLP), that switches any kind of a word to its base root mode.

In [16]:

```
# lemmetization
def lemmatized_words(text):
    lemm = nltk.stem.WordNetLemmatizer()
    lst=nltk.word_tokenize(text)
    lemmatized_output = ' '.join([lemm.lemmatize(w) for w in lst])
    return lemmatized_output

df['lemmetized_data'] = list(map(lemmatized_words, df.clean_stop))
```

Average length of reviews before preprocessing and after preprocessing

In [17]:

```
print(avg_len_reviews("cleaned_text_reviews"),avg_len_reviews("lemmetized_data"),sep=",")
```

```
183.6413,110.437
```

In this assignment, I have taken 2 approaches to evaluate the model performance on test data (20%)

1. With data preprocessing (stop words removal and lemmetization)

2. Without data preprocessing (No stop words removal and No lemmetization

I have employed all the steps and I found that the model performance - precision, recall and f1 score is better on all models without the data preprocessed data and the justification/proof of this is shown below

With data preprocessing (stop words removal and lemmetization)

Splitting Data (80%-20%)

In [18]:

```
X=df['lemmetized_data']
y=df['rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42,stratify=y)
```

4. Feature Extraction

TF-IDF Feature Extraction

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents.

In [19]:

```
vectorizer = TfidfVectorizer(ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

Function to print all metrics - precision, recall and F1-score along with the averages as per the assignment guidelines

In [20]:

```
def results(y_test,y_pred):
    precision, recall, fscore, support = score(y_test, y_pred)
    averages=score(y_test, y_pred, average='weighted')
    for i in range(len(precision)):
        print(precision[i],recall[i],fscore[i],sep=",")
    print(averages[0],averages[1],averages[2],sep=",")
```

5. Perceptron

Perceptron is a single layer neural network and a multi-layer perceptron is called Neural Networks.

In [21]:

```
perceptron_text_clf = Perceptron()
perceptron_text_clf.fit(X_train, y_train)
perceptron_predictions = perceptron_text_clf.predict(X_test)
```

Classification results for Perceptron model on test data (20%)

In [22]:

```
results(y_test,perceptron_predictions)
```

```
0.6505566801619433,0.64275,0.6466297786720322
0.560454065469905,0.53075,0.5451977401129944
0.7147887323943662,0.76125,0.7372881355932204
0.6419331593420715,0.6449166666666667,0.6430385514594156
```

6. SVM

The objective of the support vector machine algorithm is to find a hyperplane in an N-dimensional space(N — the number of features) that distinctly classifies the data points.

In [23]:

```
svc_text_clf = LinearSVC()
svc_text_clf.fit(X_train, y_train)
svc_predictions = svc_text_clf.predict(X_test)
```

Classification results for SVM (SVC) model on test data (20%)

In [24]:

```
results(y_test,svc_predictions)
```

```
0.6895693303460294,0.6925,0.6910315579393788
0.6069348861831657,0.57325,0.589611725379275
0.7531510107015458,0.79175,0.771968312004875
0.6832184090769136,0.6858333333333333,0.6842038651078428
```

7. Logistic Regression

In [25]:

```
lr_text_clf = LogisticRegression(max_iter=500)
lr_text_clf.fit(X_train, y_train)
lr_predictions = lr_text_clf.predict(X_test)
```

Classification results for Logistic Regression model on test data (20%)

In [26]:

```
results(y_test,lr_predictions)
```

```
0.7009206270216471,0.70425,0.7025813692480359
0.6181262729124236,0.607,0.6125126135216952
0.771527263755243,0.78175,0.7766049919284739
0.6968580545631047,0.6976666666666667,0.6972329915660683
```

8. Naive Bayes

In [27]:

```
mnb_text_clf = MultinomialNB()
mnb_text_clf.fit(X_train, y_train)
mnb_predictions = mnb_text_clf.predict(X_test)
```

Classification results for Multinomial Naive Bayes model on test data (20%)

In [28]:

```
results(y_test,mnb_predictions)
```

```
0.7094540612516644,0.666,0.6870406189555125
0.5885817852288174,0.6495,0.6175421915854529
0.7828243278517358,0.74975,0.7659302771038181
0.6936200581107392,0.6884166666666667,0.6901710292149279
```

Without data preprocessing

Splitting Data (80%-20%)

In [29]:

```
X=df['cleaned_text_reviews']
y=df['rating']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42,stratify=y)
```

4. Feature Extraction

TF-IDF Feature Extraction

In [30]:

```
vectorizer = TfidfVectorizer(ngram_range=(1,2))
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
```

5. Perceptron

In [31]:

```
perceptron_text_clf = Perceptron()
perceptron_text_clf.fit(X_train, y_train)
perceptron_predictions = perceptron_text_clf.predict(X_test)
```

Classification results for Perceptron model on test data (20%)

In [32]:

```
results(y_test,perceptron_predictions)
```

```
0.6838905775075987,0.73125,0.706777818050018
0.6270435023552231,0.56575,0.5948219214088579
0.7836655323286339,0.806,0.7946758688686222
0.6981998707304853,0.701,0.698758536109166
```

6. SVM

In [33]:

```
svc_text_clf = LinearSVC()
svc_text_clf.fit(X_train, y_train)
svc_predictions = svc_text_clf.predict(X_test)
```

Classification results for SVM (SVC) model on test data (20%)

In [34]:

```
results(y_test,svc_predictions)
```

```
0.72314453125,0.7405,0.731719367588933
0.6545360020795425,0.6295,0.6417739263412768
0.8237614000492975,0.8355,0.8295891771130695
0.7338139777929467,0.7351666666666666,0.7343608236810931
```

7. Logistic Regression

In [35]:

```
lr_text_clf = LogisticRegression(max_iter=500)
lr_text_clf.fit(X_train, y_train)
lr_predictions = lr_text_clf.predict(X_test)
```

Classification results for Logistic Regression model on test data (20%)

In [36]:

```
results(y_test,lr_predictions)
```

```
0.7282845390757319,0.7525,0.7401942702569777
0.6624525916561315,0.655,0.6587052168447518
0.841002044989775,0.8225,0.8316481294236603
0.7439130585738796,0.7433333333333333,0.7435158721751298
```

8. Naive Bayes

In [37]:

```
mnb_text_clf = MultinomialNB()
mnb_text_clf.fit(X_train, y_train)
mnb_predictions = mnb_text_clf.predict(X_test)
```

Classification results for Multinomial Naive Bayes model on test data (20%)

In [38]:

```
results(y_test,mnb_predictions)
```

```
0.7520443154840412,0.71275,0.7318701065331793
0.5985591354812888,0.74775,0.6648882960986996
0.8904109589041096,0.715,0.7931225734886301
0.7470048032898132,0.7251666666666666,0.729960325373503
```

References

https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Beauty_v1_00.tsv.gz

https://stackoverflow.com/questions/11331982/how-to-remove-any-url-within-a-string-in-python

https://www.geeksforgeeks.org/nlp-expand-contractions-in-text-processing/

https://stackoverflow.com/questions/33404752/removing-emojis-from-a-string-in-python

https://scikit-learn.org/stable/