NLP-HW1-Report

Aditi Hande

Importing Libraries

- Imported all the necessary libraries for the assignment
- To remove the warnings from notebook suppressed warnings by filtering

Printing Python Version

- Printing python version as asked for using python_version() from platform

Reading the dataset (amazon review tsv)

- Read the amazon_reviews_us_Beauty_v1_00 dataset which has amazon reviews of beauty products. This was done using a pandas dataframe named data.
- sep='\t',on_bad_lines='skip' this specifies that we read data which is tab separated and we skip the rows which have invalid data.
- Since we need only two columns, extracted review body and star rating in df dataframe

Balance dataset

- Defined function “classfunc” which returns class based on star rating
- Converted the column star_rating into string for uniformity and then created column class with values returned from “classfunc” using lambda func
- I had assigned class 0 to invalid rows so dropped them
- Finally sampled 2k entries from each class i.e. 1,2,3 using group by according to class column. Used sample so we have random rows selected.

Data Cleaning

- To print changed character lengths of each review before and after cleaning, calculated the average of lengths using “mean”
- Converted all reviews txt to lower case using str.lower() for uniformity
- Removed all the urls from text using regex ‘http\S+|www.\S+‘ and replaced its occurrence with “ “. the regex gives the pattern of urls
- Removed non alphabetic characters such as symbols using not form of regex for alphabets and replaced its occurrence with “ “
- Used beautifulsoup parser to remove html tags from review txt
- Used the contractions module of python to perform contractions. And joined the word list to get text again
- Printed changed character length as before and after

```
Avg length of review character before and after data-cleaning:  267.1358613217768 , 257.0709166666667
```

-

Data Cleaning (Removing Stop Words and lemmatizer)

- To print changed character lengths of each review before and after preprocessing, calculated the average of lengths using “mean”

```
Avg length of review character before and after preprocessing:  257.0709166666667 , 150.91458333333333
```

-
- To get better results we remove stop words from review text using nltk stopword module.
- We split the words in review, removed defined stopwords and joined them back for text.
- To perform lemmatization we used nltk.stem.WordNetLemmatizer.
- I defined get_pos function in which we return the identifier for pos for each word, this is later used in lemmatizer.lemmatize call (as second param) for each review text word (used simple split using whitespace for this).
- Again joined the word list to create text and Printed changed character length as before and after preprocessing

TF-IDF Feature Extraction

- To get a matrix of feature vectors of review text, used TfidfVectorizer module from sklearn.
- Model is built with parameters sublinear_tf=True,ngram_range=(1, 3),binary=True. Used this to include trigrams, bigrams and unigrams and tf is logarithmic. This aids in better feature extraction
- Saved the generated features in vectorized_x using tfidf.fit_transform() this fits parameters on data
- Later stratified split (to maintain balance) the data into test(20 percent) train (80percent) and a fixed random state for consistency as seed value will remain constant for split each time. This was done using test_size=0.2, stratify=y, random_state = 44

Perceptron

- Used sklearn module for inbuilt model of perceptron with random state as 42 (could be any feasible value). This was done using Perceptron(random_state=42)
- Fit the model on training data and later predicted the y_hat using the .predict() function.
- Used the actual y values and yhat (predicted) values to calculate recall precision and f1 scores using the sklearn.metrics.

- Extracted the per class values from the generated metrics printed those and for the average calculated the metric again with parameter specifying that average should be weighted.
- The results for the prediction are as follows.

```
Results from Perceptron model
precision, recall, f1score for class 1:  0.6476145488899386 ,  0.6855 ,  0.6660189458343455
precision, recall, f1score for class 2:  0.5645968953430145 ,  0.56375 ,  0.5641731298473855
precision, recall, f1score for class 3:  0.7417815482502651 ,  0.6995 ,  0.7200205867215645
precision, recall, f1score   average  :  0.651330997494406 ,  0.6495833333333333 ,  0.6500708874677652
```

-

SVM

- Used sklearn module for inbuilt model of SVM with random state as 42 (could be any feasible value). This was done using LinearSVC(random_state=42)
- Fit the model on training data and later predicted the y_hat using the .predict() function.
- Used the actual y values and yhat (predicted) values to calculate recall precision and f1 scores using the sklearn.metrics.
- Extracted the per class values from the generated metrics printed those and for the average calculated the metric again with parameter specifying that the average should be weighted.
- The results for the prediction is as follows.

```
Results from SVM model
precision, recall, f1score for class 1:  0.6988595001213298 ,  0.72 ,  0.7092722571111932
precision, recall, f1score for class 2:  0.6115394841797395 ,  0.575 ,  0.592707125370442
precision, recall, f1score for class 3:  0.7518212724623604 ,  0.774 ,  0.762749445676275
precision, recall, f1score   average  :  0.6874067522544764 ,  0.6896666666666667 ,  0.6882429427193034
```

-

Logistic regression

- Used sklearn module for inbuilt model of logistic regression with random state as 42 (could be any feasible value). This was done using LogisticRegression(random_state=42)
- Fit the model on training data and later predicted the y_hat using the .predict() function.
- Used the actual y values and yhat (predicted) values to calculate recall precision and f1 scores using the sklearn.metrics.
- Extracted the per class values from the generated metrics printed those and for the average calculated the metric again with parameter specifying that the average should be weighted.

- The results for the prediction are as follows.

```
Results from Logistic Regression model
precision, recall, f1score for class 1:  0.6949927780452576 ,  0.72175 ,  0.7081187147412312
precision, recall, f1score for class 2:  0.6090680100755668 ,  0.6045 ,  0.6067754077791719
precision, recall, f1score for class 3:  0.7724458204334366 ,  0.7485 ,  0.7602844083291013
precision, recall, f1score   average  :  0.6921688695180869 ,  0.6915833333333333 ,  0.6917261769498348
```

-

Naive Bayes

- Used sklearn module for inbuilt model of Naive Bayes This was done using MultinomialNB()
- Fit the model on training data and later predicted the y_hat using the .predict() function.
- Used the actual y values and yhat (predicted) values to calculate recall precision and f1 scores using the sklearn.metrics.
- Extracted the per class values from the generated metrics printed those and for the average calculated the metric again with parameter specifying that the average should be weighted.
- The results for the prediction are as follows.

```
Results from Multinomial Naive Bayes model
precision, recall, f1score for class 1:  0.720514841082217 ,  0.68575 ,  0.7027027027027027
precision, recall, f1score for class 2:  0.5966183574879227 ,  0.6175 ,  0.6068796068796068
precision, recall, f1score for class 3:  0.7508018751542067 ,  0.76075 ,  0.7557432012914441
precision, recall, f1score   average  :  0.6893116912414489 ,  0.688 ,  0.6884418369579179
```

-

References:

https://stackoverflow.com/questions/58829730/setting-the-values-of-a-pandas-df-column-based-on-ranges-of-values-of-another-df
For class creation

https://stackoverflow.com/questions/46241120/how-to-remove-non-alpha-numeric-characters-from-strings-within-a-dataframe-colum
Remove non alphabetic characters

https://stackoverflow.com/questions/51994254/removing-url-from-a-column-in-pandas-dataframe https://stackoverflow.com/questions/44703945/pandas-trouble-stripping-html-tags-from-dataframe-column

https://www.geeksforgeeks.org/pandas-strip-whitespace-from-entire-dataframe/

https://towardsdatascience.com/preprocessing-text-data-using-python-576206753c28

https://www.machinelearningplus.com/nlp/lemmatization-examples-python/

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html

https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/

https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.MultinomialNB.html

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html

In [1]:
```
import warnings
warnings.filterwarnings("ignore")

import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('omw-1.4')
from nltk.corpus import wordnet
from nltk.stem.wordnet import WordNetLemmatizer
import re
from sklearn.metrics import f1_score,recall_score,precision_score
from bs4 import BeautifulSoup
import contractions
```

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

In [2]:
```
! pip install bs4

from platform import python_version
print(python_version())

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

```
Requirement already satisfied: bs4 in c:\users\dipal\appdata\local\programs\python\python311\lib\site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\dipal\appdata\local\programs\python\python311\lib\site-packages (f
rom bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\dipal\appdata\local\programs\python\python311\lib\site-packages (fr
om beautifulsoup4->bs4) (2.3.2.post1)
3.11.1
```

Read Data

In [3]:
```
data=pd.read_csv("amazon_reviews_us_Beauty_v1_00.tsv",sep='\t',on_bad_lines='skip')
```

Keep Reviews and Ratings

In [4]:
```
df=data.loc[:,["review_body","star_rating"]]
```

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

In [5]:
```
def classfunc(star_):
    if star_=='5'or star_=='4':
        return 3
    elif star_=='3':
        return 2
    elif star_ =='2'or star_ =='1':
        return 1
    else:
        return 0

df['star_rating'] = df['star_rating'].astype(str)
df['class'] = df['star_rating'].apply(lambda x: classfunc(x[0]))
df.drop(df[(df['class'] == 0)].index, inplace=True) #dropping all entries with incorrect data

df_balanced = pd.DataFrame()
df_balanced = df.groupby(["class"]).apply(lambda grp: grp.sample(n=20000)) #selecting 2k entries from all three classes
```

Data Cleaning

Pre-processing

In [6]:
```
before_datacleaning_len = df_balanced['review_body'].str.len().mean()

df_balanced['review_body'] = df_balanced['review_body'].str.lower()
df_balanced['review_body'] = df_balanced['review_body'].str.replace('http\S+|www.\S+', '', case=False)
df_balanced['review_body'] = df_balanced['review_body'].str.replace('[^a-zA-Z ]', '')
X=df_balanced.review_body
df_balanced['review_body'] = [BeautifulSoup(X).get_text() for X in df_balanced['review_body'].astype(str) ]
df_balanced['review_body'] = df_balanced['review_body'].str.strip()

df_balanced['review_body'] = df_balanced['review_body'].apply(lambda x: [contractions.fix(word) for word in x.split()])
df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]

after_datacleaning_len = df_balanced['review_body'].str.len().mean()


print("Avg length of review character before and after data-cleaning: ", before_datacleaning_len,",",after_datacleaning_len)
```

```
Avg length of review character before and after data-cleaning:  267.1358613217768 , 257.0709166666667
```

remove the stop words

In [7]:
```
before_preprocessing_len = df_balanced['review_body'].str.len().mean()

from nltk.corpus import stopwords

#stop words removal
english_stopwords = stopwords.words('english')
#df_balanced['review_body'] = [t for t in tokens if t not in english_stopwords]
df_balanced['review_body']= df_balanced['review_body'].apply(lambda x: [item for item in x.split() if item not in english_sto
df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]
```

perform lemmatization

In [8]:
```
#Lemmatization

lemmatizer = nltk.stem.WordNetLemmatizer()


def get_pos(word):
  tag = nltk.pos_tag([word])[0][1][0].upper()
  tag_dict = {"J": wordnet.ADJ,
              "N": wordnet.NOUN,
              "V": wordnet.VERB,
              "R": wordnet.ADV}

  return tag_dict.get(tag, wordnet.NOUN)

df_balanced['review_body'] = df_balanced['review_body'].apply(lambda x: [lemmatizer.lemmatize(w, get_pos(w)) for w in x.split
df_balanced['review_body'] = [' '.join(map(str, word)) for word in df_balanced['review_body']]

after_preprocessing_len = df_balanced['review_body'].str.len().mean()



print("Avg length of review character before and after preprocessing: ",before_preprocessing_len, ",", after_preprocessing_le
```

```
Avg length of review character before and after preprocessing:  257.0709166666667 , 150.91458333333333
```

TF-IDF Feature Extraction

In [9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split


tfidf = TfidfVectorizer(sublinear_tf=True,ngram_range=(1, 3),binary=True)


x = df_balanced['review_body']
y = df_balanced['class']

vectorized_x = tfidf.fit_transform(x)


X_train, X_test, y_train, y_test = train_test_split(vectorized_x, y,test_size=0.2, stratify=y, random_state = 44)
```

Perceptron

In [10]:
```
from sklearn.utils import multiclass
from sklearn.linear_model import Perceptron
from sklearn.metrics import classification_report


model_perceptron = Perceptron(random_state=42)
model_perceptron.fit(X_train, y_train)

y_hat = model_perceptron.predict(X_test)

precision_perceptron = precision_score(y_test,y_hat,average=None)
recall_perceptron = recall_score(y_test,y_hat,average=None)
f1_perceptron = f1_score(y_test,y_hat,average=None)

print("Results from Perceptron model")

print("precision, recall, f1score for class 1: ", precision_perceptron[0], ", ", recall_perceptron[0], ", ", f1_perceptron[0]
print("precision, recall, f1score for class 2: ", precision_perceptron[1], ", ", recall_perceptron[1], ", ", f1_perceptron[1]
print("precision, recall, f1score for class 3: ", precision_perceptron[2], ", ", recall_perceptron[2], ", ", f1_perceptron[2]
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", recall_score(y_test



#print(classification_report(model_perceptron.predict(X_test), y_test))
```

```
Results from Perceptron model
precision, recall, f1score for class 1:  0.6476145488899386 ,  0.6855 ,  0.6660189458343455
precision, recall, f1score for class 2:  0.5645968953430145 ,  0.56375 ,  0.5641731298473855
precision, recall, f1score for class 3:  0.7417815482502651 ,  0.6995 ,  0.7200205867215645
precision, recall, f1score   average  :  0.651330997494406 ,  0.6495833333333333 ,  0.6500708874677652
```

SVM

In [11]:
```
from sklearn.svm import LinearSVC

model_svc = LinearSVC(random_state=42)
model_svc.fit(X_train, y_train)

y_hat = model_svc.predict(X_test)

precision_svc = precision_score(y_test,y_hat,average=None)
recall_svc = recall_score(y_test,y_hat,average=None)
f1_svc = f1_score(y_test,y_hat,average=None)

print("Results from SVM model")


print("precision, recall, f1score for class 1: ", precision_svc[0], ", ", recall_svc[0], ", ", f1_svc[0])
print("precision, recall, f1score for class 2: ", precision_svc[1], ", ", recall_svc[1], ", ", f1_svc[1])
print("precision, recall, f1score for class 3: ", precision_svc[2], ", ", recall_svc[2], ", ", f1_svc[2])
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", recall_score(y_test


#print(classification_report(model_svc.predict(X_test), y_test))
```

```
Results from SVM model
precision, recall, f1score for class 1:  0.6988595001213298 ,  0.72 ,  0.7092722571111932
precision, recall, f1score for class 2:  0.6115394841797395 ,  0.575 ,  0.592707125370442
precision, recall, f1score for class 3:  0.7518212724623604 ,  0.774 ,  0.762749445676275
precision, recall, f1score   average  :  0.6874067522544764 ,  0.6896666666666667 ,  0.6882429427193034
```

Logistic Regression

In [12]:
```
from sklearn.linear_model import LogisticRegression

model_logistic = LogisticRegression(random_state=42)
model_logistic.fit(X_train, y_train)

y_hat = model_logistic.predict(X_test)

precision_logistic = precision_score(y_test,y_hat,average=None)
recall_logistic = recall_score(y_test,y_hat,average=None)
f1_logistic = f1_score(y_test,y_hat,average=None)

print("Results from Logistic Regression model")


print("precision, recall, f1score for class 1: ", precision_logistic[0], ", ", recall_logistic[0], ", ", f1_logistic[0])
print("precision, recall, f1score for class 2: ", precision_logistic[1], ", ", recall_logistic[1], ", ", f1_logistic[1])
print("precision, recall, f1score for class 3: ", precision_logistic[2], ", ", recall_logistic[2], ", ", f1_logistic[2])
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", recall_score(y_test


#print(classification_report(model_logistic.predict(X_test), y_test))
```

```
Results from Logistic Regression model
precision, recall, f1score for class 1:  0.6949927780452576 ,  0.72175 ,  0.7081187147412312
precision, recall, f1score for class 2:  0.6090680100755668 ,  0.6045 ,  0.6067754077791719
precision, recall, f1score for class 3:  0.7724458204334366 ,  0.7485 ,  0.7602844083291013
precision, recall, f1score   average  :  0.6921688695180869 ,  0.6915833333333333 ,  0.6917261769498348
```

Naive Bayes

In [13]:
```
from sklearn.naive_bayes import MultinomialNB

model_mnb = MultinomialNB()
model_mnb.fit(X_train,y_train)

y_hat = model_mnb.predict(X_test)

precision_mnb = precision_score(y_test,y_hat,average=None)
recall_mnb = recall_score(y_test,y_hat,average=None)
f1_mnb = f1_score(y_test,y_hat,average=None)

print("Results from Multinomial Naive Bayes model")


print("precision, recall, f1score for class 1: ", precision_mnb[0], ", ", recall_mnb[0], ", ", f1_mnb[0])
print("precision, recall, f1score for class 2: ", precision_mnb[1], ", ", recall_mnb[1], ", ", f1_mnb[1])
print("precision, recall, f1score for class 3: ", precision_mnb[2], ", ", recall_mnb[2], ", ", f1_mnb[2])
print("precision, recall, f1score   average  : ", precision_score(y_test,y_hat,average='weighted'), ", ", recall_score(y_test


#print(classification_report(model_mnb.predict(X_test), y_test))
```

```
Results from Multinomial Naive Bayes model
precision, recall, f1score for class 1:  0.720514841082217 ,  0.68575 ,  0.7027027027027027
precision, recall, f1score for class 2:  0.5966183574879227 ,  0.6175 ,  0.6068796068796068
precision, recall, f1score for class 3:  0.7508018751542067 ,  0.76075 ,  0.7557432012914441
precision, recall, f1score   average  :  0.6893116912414489 ,  0.688 ,  0.6884418369579179
```