CSCI544: Homework Assignment No1

Samhitha Gundam

1866055918

1. Downloaded the dataset from provided URL and loaded using pandas read_csv() function and considered only reviews and rating columns needed as part of this assignment.

2. Data cleaning - Using the below data cleaning methods cleaned the data. The average length of the reviews before and after cleaning is 189.51892,182.91149

    1. Dropped the null value rows

    2. The ratings column had few float and few int values, converted all of them to int and later grouped by class and considered only 20,000 samples of each class

    3. Further cleaned the data by converting all reviews to lower case. Next expanded all the contractions. Removed the urls from the reviews data. Removed all the characters from reviews that are not alphabets numbers or space.

3. Used NLTK stop words list to remove the stop words from reviews. Also used WordNetLemmatizer for lemmatizing the data. Average count before and after this preprocessing is 182.91149,110.65683

4. Next performed TF-IDF on the review data to create features from the language dataset that can further used in different models that can help classify the data. Using this created the features(X) and labels(Y) needed for training models. Split the data into 80 and 20% to get train and test data.

5. Perceptron model - used sklearn built in perceptron model. Ran it for 1000 iterations. Increase after 1000 iterations did not bring much change in score metrics. Scores using perceptron model are as follows-

	Precision	Recall	F1-score
Class 1	0.499	0.45	0.473
Class 2	0.311	0.364	0.336
Class 3	0.317	0.265	0.289
Class 4	0.362	0.392	0.376
Class 5	0.55	0.557	0.554
Average	0.407	0.405	0.405

6. SVM - used sklearn built in SVM(linearSVC) model. Scores using SVM model are as follows -

	Precision	Recall	F1-score
Class 1	0.542	0.642	0.588
Class 2	0.374	0.328	0.350
Class 3	0.402	0.334	0.365
Class 4	0.431	0.406	0.418
Class 5	0.610	0.711	0.656
Average	0.471	0.483	0.474

7. Logistic Regression- used sklearn built in Logistic Regression model. Ran it for 1000 iterations. Increase after 1000 iterations did not bring much change in score metrics. Score using Logistic Regression as follows-

	Precision	Recall	F1-score
Class 1	0.575	0.642	0.607
Class 2	0.410	0.385	0.397
Class 3	0.421	0.392	0.406
Class 4	0.465	0.442	0.453
Class 5	0.649	0.693	0.670
Average	0.504	0.510	0.506

8. Multinomial Naive Bayes - used sklearn built in MultinomialNB() model. Scores using this model are as follows-

	Precision	Recall	F1-score
Class 1	0.583	0.594	0.588
Class 2	0.391	0.389	0.390
Class 3	0.400	0.392	0.396
Class 4	0.444	0.439	0.441
Class 5	0.653	0.662	0.658
Average	0.493	0.494	0.494

In [1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('omw-1.4')
import re
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings("ignore")
```

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

Read Data

In [2]:
```
raw_data=pd.read_csv("amazon_reviews_us_Jewelry_v1_00.tsv",sep="\t",on_ba
```

Keep Reviews and Ratings

In [3]:
```
 data=raw_data[['star_rating','review_body']]
```

In [4]:
```
data=data.dropna()
data = data.reset_index(drop=True)
data['star_rating']=data['star_rating'].astype(int)
```

We select 20000 reviews randomly from each rating class.

In [5]:
```
df=data.groupby('star_rating').sample(n=20000)
```

Data Cleaning

In [6]:
```
cnt_before=(df['review_body'].str.len()).mean()
#print("Average review_body character count before  Datcleaning:"+ str(st
```

In [7]:
```
df['review_body']=df['review_body'].str.lower()  #convert to lower case

#remove contractions
df['review_body']=df['review_body'].str.replace("\'re"," are")
df['review_body']=df['review_body'].str.replace("br"," ")
df['review_body']=df['review_body'].str.replace("\n't"," not")
df['review_body']=df['review_body'].str.replace("\'s'"," is")
df['review_body']=df['review_body'].str.replace("i'm"," i am")
df['review_body']=df['review_body'].str.replace("\'ve'"," have")
df['review_body']=df['review_body'].str.replace("din't","did not")

df['review_body']=df['review_body'].str.replace('http\S+|www.\S+', '', ca
df['review_body']=df['review_body'].str.replace('[^a-zA-Z0-9 ]', '')
df['review_body']=df['review_body'].str.replace('/ +/', ' ')#convert mult
df['review_body']=df['review_body'].str.replace('/^ /', '') # remove spac
df['review_body']=df['review_body'].str.replace('/ $/', '') # remove unne
```

In [8]:
```
cnt_after=(df['review_body'].str.len()).mean()
print(str(cnt_before)+","+str(cnt_after))
#print("Average review_body character count after Data cleaning:"+ str(st
```

```
189.51892,182.91149
```

Pre-processing

In [9]:
```
cnt_before=(df['review_body'].str.len()).mean()
#print("Average review_body character count before preprocessing:"+ str(s
```

remove the stop words

In [10]:
```
from nltk.corpus import stopwords

stop_words=stopwords.words('english')
df['review_body']=df['review_body'].apply(lambda x: ' '.join([word for wo
```

perform lemmatization

In [11]:
```
from nltk.stem import WordNetLemmatizer
lemmatizer = nltk.stem.WordNetLemmatizer()
df['review_body']=df['review_body'].apply(lambda x:' '.join([lemmatizer.l
```

In [12]:
```
cnt_after=(df['review_body'].str.len()).mean()
#print("Average review_body character count after preprocessing:"+ str(st
print(str(cnt_before)+","+str(cnt_after))
```

```
182.91149,110.65683
```

TF-IDF Feature Extraction

In [13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
vectorizer= TfidfVectorizer()
X=df['review_body']
X=vectorizer.fit_transform(X)
Y=df['star_rating']
```

In [14]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train, Y_test = train_test_split(X,Y, test_size=0.2, ran
```

Perceptron

In [15]:
```
from sklearn.linear_model import Perceptron
clf_perceptron = Perceptron()
clf_perceptron.fit(X_train, Y_train)
```

Out[15]:
```
Perceptron()
```

In [16]:
```
y_test_perceptron=clf_perceptron.predict(X_test)
```

In [17]:
```
from sklearn.metrics import classification_report
report=classification_report(Y_test, y_test_perceptron,output_dict=True)
```

In [18]:
```
print(str(report['1']['precision'])+","+str(report['1']['recall'])+","+st
print(str(report['2']['precision'])+","+str(report['2']['recall'])+","+st
print(str(report['3']['precision'])+","+str(report['3']['recall'])+","+st
print(str(report['4']['precision'])+","+str(report['4']['recall'])+","+st
print(str(report['5']['precision'])+","+str(report['5']['recall'])+","+st
print(str(report['weighted avg']['precision'])+","+str(report['weighted a
```

```
0.49903448275862067,0.45044820717131473,0.47349823321554774
0.3118991668446913,0.3639990027424582,0.3359410952600092
0.3171385991058122,0.2646107933349913,0.2885032537960955
0.3616727941176471,0.3917371826779492,0.37610513739546
0.5500376222723853,0.5574478901881037,0.5537179649034213
0.4073706820380709,0.405,0.4049308568590327
```

SVM

In [19]:
```
from sklearn.svm import LinearSVC
clf_SVM = LinearSVC()
clf_SVM.fit(X_train, Y_train)
y_test_SVM=clf_SVM.predict(X_test)
```

In [20]:
```
report_SVM=classification_report(Y_test, y_test_SVM,output_dict=True)
```

In [21]:
```
print(str(report_SVM['1']['precision'])+","+str(report_SVM['1']['recall']
print(str(report_SVM['2']['precision'])+","+str(report_SVM['2']['recall']
print(str(report_SVM['3']['precision'])+","+str(report_SVM['3']['recall']
print(str(report_SVM['4']['precision'])+","+str(report_SVM['4']['recall']
print(str(report_SVM['5']['precision'])+","+str(report_SVM['5']['recall']
print(str(report_SVM['weighted avg']['precision'])+","+str(report_SVM['we
```

```
0.5417891642167156,0.6424302788844621,0.5878332194121668
0.37404146549275774,0.32834704562453254,0.34970791290493897
0.4024572969733293,0.3339965182790351,0.3650448491437891
0.4305702217529039,0.4059233449477352,0.41788367922111197
0.609102787456446,0.7109811896288765,0.6561107201501291
0.4710313954786182,0.4834,0.4745729091204592
```

Logistic Regression

In [22]:
```
from sklearn.linear_model import LogisticRegression
clf_logistic = LogisticRegression(max_iter=1000,solver='lbfgs')
clf_logistic.fit(X_train,Y_train)
y_test_logistic=clf_logistic.predict(X_test)
```

In [23]:
```
report_logistic=classification_report(Y_test, y_test_logistic,output_dict
```

In [24]:
```
print(str(report_logistic['1']['precision'])+","+str(report_logistic['1']
print(str(report_logistic['2']['precision'])+","+str(report_logistic['2']
print(str(report_logistic['3']['precision'])+","+str(report_logistic['3']
print(str(report_logistic['4']['precision'])+","+str(report_logistic['4']
print(str(report_logistic['5']['precision'])+","+str(report_logistic['5']
print(str(report_logistic['weighted avg']['precision'])+","+str(report_lo
```

```
0.5755749051127483,0.6419322709163346,0.6069452619187757
0.41004517672070157,0.3846920967339815,0.3969642397736043
0.42112299465240643,0.3916936085550858,0.4058755315036722
0.46551271964332547,0.4417620706819313,0.45332652279402375
0.6487514863258026,0.6934417895271988,0.67035262317238
0.5036077019494722,0.50995,0.5060187218960263
```

Naive Bayes

In [25]:
```
from sklearn.naive_bayes import MultinomialNB
clf_NB = MultinomialNB()
clf_NB.fit(X_train,Y_train)
y_test_NB= clf_NB.predict(X_test)
```

In [26]:
```
report_NB=classification_report(Y_test, y_test_NB,output_dict=True)
```

In [27]:
```
print(str(report_NB['1']['precision'])+","+str(report_NB['1']['recall'])+
print(str(report_NB['2']['precision'])+","+str(report_NB['2']['recall'])+
print(str(report_NB['3']['precision'])+","+str(report_NB['3']['recall'])+
print(str(report_NB['4']['precision'])+","+str(report_NB['4']['recall'])+
print(str(report_NB['5']['precision'])+","+str(report_NB['5']['recall'])+
print(str(report_NB['weighted avg']['precision'])+","+str(report_NB['weig
```

```
0.5833129434793247,0.5936254980079682,0.5884240404788351
0.39098873591989985,0.3894290700573423,0.39020734449163125
0.3992914979757085,0.3924396916190003,0.39583594631882607
0.44388525415198793,0.43902439024390244,0.44144144144144143
0.6530561122244489,0.6626842907981698,0.6578349735049206
0.493452270541084,0.49475,0.49407617214735017
```