NLP HW1 Report

- Indra Kumar Vijaykumar (2676313551)

Libraries used :

- Pandas
- Numpy
- NLTK
  - Stopwords
  - PorterStemmer
  - WordNetLemmatizer
- Regex (re)
- BeautifulSoup
- Contractions
- Sklearn
  - TfidfVectorizer
  - train_test_split
  - Perceptron
  - MultinomialNB
  - LinearSVC
  - LogisticRegression

Dataset: Amazon Reviews Dataset for the jewelry category (https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Jewelry_v1_00.tsv.gz)

Read Data:
Made use of the Pandas library to read the data from the tsv file with additional parameters:
"sep": set to "\t" as it is a tab separated file
"error_bad_lines" : set to "False" to skip lines that can cause and error while reading the dataset

Keep Reviews and Ratings:
Reduced the entire dataframe to only two columns, "review_body" and "star_rating" where the review_body column is our input column and the star_rating column is our target column.

Select 20,000 random rows under each rating class:
Used the df.sample() function in the pandas library to sample **"50,000"** random rows from the data frame under each rating. 50,000 rows were selected instead of 20,000 rows as after the cleaning step is applied, certain rows lose all their input text to the cleaning process and become empty. Having additional rows accounts for this loss and before the Tf-IDF step, these 50,000 rows under each class are reduced down to 20,000 rows under each rating class.

Data Cleaning: Regex is used to perform all the following cleaning operations on the data.

- **Removing HTML tags:** The regular expression '<.*?>' is used to remove all the html tags in the input text by replacing the matched characters with “”.
- **Removing URLs:** The regular expression 'http\S+' is used to remove all the urls in the input text by replacing the matched characters with “”.
- **Removing non-alphabetic characters:** The regular expression '[^A-Za-z]' is used to remove all the non-alphabetic characters other than whitespace in the input text by replacing the matched characters with “”.
- **Reducing multiple consecutive whitespace characters to a single whitespace:** The regular expression ' +' is used to reduce multiple consecutive whitespace characters to a single whitespace character in the input text.

Data Preprocessing

- **Stemming:** The Porterstemmer function from the nltk library is used to apply the process of stemming on the input text
- **Lemmatization:** The WordNetLemmatizer function from the nltk library is used to apply the process of lemmatization to the input text.

Note: It is during this lemmatization process that the data under each category is reduced to 20,000 from 50,000 and only lines with at least 13 words are included in the input data. The value 13 was arrived at via trial and error between 10 to 15.

Feature extraction using TF-IDF vectorization

- The dataset size at this point is 100,000 samples with 20,000 samples under each of the 5 star_rating categories.
- TF-IDF Vectorization using the Tfidfvectorizer function from the sklearn library is applied to this data to convert it into a sparse matrix format which the machine learning models can understand.

Train Test split : The dataset is split into 80% training data and 20% testing data using the train_test_split function from the sklearn library.

Model Training

Perceptron

- The perceptron function from the sklearn library is used to implement the perceptron model. Sample results from this model are as follows:

```
perceptron report:
              precision    recall  f1-score   support

           1       0.54      0.50      0.52      4081
           2       0.32      0.33      0.32      4029
           3       0.30      0.33      0.32      3965
           4       0.37      0.29      0.32      4015
           5       0.54      0.65      0.59      3910

    accuracy                           0.42     20000
   macro avg       0.42      0.42      0.42     20000
weighted avg       0.42      0.42      0.41     20000
```

SVM

- The linear_svc function from the sklearn library is used to implement the SVM model. Sample results from this model are as follows:

```
svm report:
              precision    recall  f1-score   support

           1       0.57      0.67      0.62      4081
           2       0.39      0.32      0.36      4029
           3       0.39      0.34      0.36      3965
           4       0.46      0.43      0.44      4015
           5       0.60      0.74      0.66      3910

    accuracy                           0.50     20000
   macro avg       0.48      0.50      0.49     20000
weighted avg       0.48      0.50      0.49     20000
```

Logistic Regression

- The logistic regression function from the sklearn library is used to implement the logistic regression model. Sample results from this model are as follows:

```
logistic regression report:
              precision    recall  f1-score   support

           1       0.61      0.66      0.63      4081
           2       0.42      0.38      0.40      4029
           3       0.41      0.39      0.40      3965
           4       0.49      0.46      0.48      4015
           5       0.65      0.72      0.68      3910

    accuracy                           0.52     20000
   macro avg       0.51      0.52      0.52     20000
weighted avg       0.51      0.52      0.52     20000
```

Multinomial Naive Bayes

- The Multinomial Naive Bayes function from the sklearn library is used to implement the naive bayes model. Sample results from this model are as follows:

```
naive bayes model report:
              precision    recall  f1-score   support

           1       0.61      0.61      0.61      4081
           2       0.41      0.38      0.39      4029
           3       0.40      0.40      0.40      3965
           4       0.46      0.44      0.45      4015
           5       0.63      0.68      0.65      3910

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.50     20000
weighted avg       0.50      0.50      0.50     20000
```

In [1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
```

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

In [2]:
```
# ! pip install bs4 # in case you don't have it installed

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

Read Data

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

```
G:\Pytorch_practice\lib\site-packages\IPython\core\interactiveshell.py:3457: Dt
ypeWarning: Columns (7) have mixed types.Specify dtype option on import or set
low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Keep Reviews and Ratings

In [4]:
```
required_columns=['review_body','star_rating']
df=org_df[required_columns]
df.head()
```

Out[4]:

	review_body	star_rating
0	so beautiful even tho clearly not high end	5
1	Great product.. I got this set for my mother, ...	5
2	Exactly as pictured and my daughter's friend l...	5
3	Love it. Fits great. Super comfortable and nea...	5
4	Got this as a Mother's Day gift for my Mom and...	5

We select 20000 reviews randomly from each rating class.

Selecting 50,000 reviews per class to account for possible loss of complete input text value in certain rows during the data cleaning process. Inputs are later reduced to 20,000 per class before the tf-idf vectorization step

In [5]:
```
import random
new_df=pd.DataFrame({"review_body":[],"star_rating":[]})
for i in range(1,6):
    new_df=pd.concat([new_df,df.loc[df['star_rating']==i].sample(50000)])
new_df.head()
```

Out[5]:

	review_body	star_rating
1166407	I returned it for the broken clasp, but the br...	1
1582207	The look of the item is great! The quality of ...	1.0
1069996	One of the lockets came off of the cylinder, i...	1
481279	Not as pretty or feminine as I had hoped. Kin...	1
438448	My ear was too big for these and I have a pret...	1

Data Cleaning

Pre-processing

In [6]:
```
charcount_init=0
charcount_final=0
str(new_df['review_body'].iloc[0])
```

Out[6]:
```
'I returned it for the broken clasp, but the braclet is TINY.  Can only imagine wearing it as one of many, many on my wrist at a time.'
```

In [7]:
```
import re
remove_html_tags='<.*?>';
remove_urls='http\S+';
remove_non_alpha='[^A-Za-z ]'
remove_extra_space=' +'
processed={"review_body":[],"star_rating":[]}
for i in range(len(new_df)):
    s=str(new_df['review_body'].iloc[i])
    c=new_df['star_rating'].iloc[i]
    charcount_init+=len(s)
    if s=="":
        continue
    s=re.sub(remove_html_tags,"",s)
    s=re.sub(remove_urls,"",s)
    s=re.sub(remove_non_alpha,"",s)
    s=re.sub(remove_extra_space," ",s)
    if s=="":
        continue
    processed["review_body"].append(contractions.fix(s.lower()))
    charcount_final+=len(processed["review_body"][-1])
    processed["star_rating"].append(int(c))
```

In [8]:
```
processed["review_body"][0]
```

Out[8]:
```
'i returned it for the broken clasp but the braclet is tiny can only imagine we
aring it as one of many many on my wrist at a time'
```

In [9]:
```
#Count before cleaning and after cleaning
print(charcount_init,charcount_final)
```

```
47033946 45171043
```

remove the stop words

In [10]:
```
from nltk.corpus import stopwords
nltk.download("stopwords")
stop_words = set(stopwords.words('english'))
pre_processcount=0

print("with stop words:",processed['review_body'][5001])
for i in range(len(processed['review_body'])):
    s=processed['review_body'][i]
    pre_processcount+=len(s)
    s=s.split(" ")
    s=[word for word in s if word not in stop_words]
    s=" ".join(s)
    processed['review_body'][i]=s
print()
print("without stop words:",processed['review_body'][5001])
```

```
with stop words: way too small

[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\indra\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!


without stop words: way small
```

Perform Stemming

In [11]:
```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
pre_stemmcount=0
post_stemmcount=0
print("before stemming: ",processed['review_body'][5001])
for i in range(len(processed['review_body'])):
    s=processed['review_body'][i]
    pre_stemmcount+=len(s)
    s=s.split(" ")
    s=[ps.stem(word) for word in s]
    s=" ".join(s)
    post_stemmcount+=len(s)
    processed["review_body"][i]=s
print("after stemming: ",processed['review_body'][5001])
```

```
before stemming:  way small
after stemming:  way small
```

perform lemmatization

In [12]:
```
from nltk.stem import WordNetLemmatizer
# nltk.download('omw-1.4')
lemmatizer = WordNetLemmatizer()
post_processcount=0

print("without lemmatization:",processed['review_body'][0])

count={1:0,2:0,3:0,4:0,5:0}
final_processed_text={"input":[],"target":[]}
for i in range(len(processed['review_body'])):
    s=processed['review_body'][i]
    c=processed['star_rating'][i]
    if count[int(c)]>=20000:
        continue
    s=s.split(" ")
    s=[lemmatizer.lemmatize(word) for word in s]
    if len(s)<13:
        continue
    s=" ".join(s)
    post_processcount+=len(s)
    final_processed_text["input"].append(s)
    final_processed_text["target"].append(int(c))
    count[int(c)]+=1
print("Before processing and after processing count",pre_processcount,post_proces
print("with lemmatization:",processed['review_body'][0])
```

```
without lemmatization: return broken clasp braclet tini imagin wear one mani ma
ni wrist time
Before processing and after processing count 45171043 16115424
with lemmatization: return broken clasp braclet tini imagin wear one mani mani
wrist time
```

In [13]:
```
print(count)
```

```
{1: 20000, 2: 20000, 3: 20000, 4: 20000, 5: 20000}
```

TF-IDF Feature Extraction

In [14]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
vector_rep=tfidf.fit_transform(final_processed_text['input'])
```

In [15]:
```
from sklearn.model_selection import train_test_split

Xtrain,Xtest,ytrain,ytest=train_test_split(vector_rep,final_processed_text['targe
print("Xtrain: ",Xtrain.shape)
print("ytrain: ",len(ytrain))
print("Xtest: ",Xtest.shape)
print("ytest: ",len(ytest))
```

```
Xtrain:  (80000, 51898)
ytrain:  80000
Xtest:  (20000, 51898)
ytest:  20000
```

Perceptron

In [16]:
```
from sklearn.linear_model import Perceptron

perceptron=Perceptron()
perceptron.fit(Xtrain,ytrain)
perceptron.score(Xtest,ytest)
```

Out[16]: 0.41605

SVM

In [17]:
```
from sklearn.svm import LinearSVC
svm=LinearSVC()
svm.fit(Xtrain,ytrain)
svm.score(Xtest,ytest)
```

Out[17]: 0.4992

Logistic Regression

In [18]:
```
from sklearn.linear_model import LogisticRegression
log_reg=LogisticRegression(max_iter=400)
log_reg.fit(Xtrain,ytrain)
log_reg.score(Xtest,ytest)
```

Out[18]: 0.52125

Naive Bayes

In [19]:
```
from sklearn.naive_bayes import MultinomialNB

nb=MultinomialNB()
nb.fit(Xtrain,ytrain)
nb.score(Xtest,ytest)
```

Out[19]: 0.4998

Create classification reports for each model

In [20]:
```
from sklearn.metrics import classification_report
perceptron_pred=perceptron.predict(Xtest)
svmpred=svm.predict(Xtest)
log_reg_pred=log_reg.predict(Xtest)
nb_pred=nb.predict(Xtest)

percept=classification_report(ytest,perceptron_pred)
svm_rep=classification_report(ytest,svmpred)
log_reg_rep=classification_report(ytest,log_reg_pred)
nb_rep=classification_report(ytest,nb_pred)

print("perceptron report:")
print(percept)
print("svm report:")
print(svm_rep)
print("logistic regression report:")
print(log_reg_rep)
print("naive bayes model report:")
print(nb_rep)
```

```
perceptron report:
              precision    recall  f1-score   support

           1       0.52      0.52      0.52      4081
           2       0.33      0.27      0.30      4029
           3       0.31      0.33      0.32      3965
           4       0.37      0.41      0.39      4015
           5       0.55      0.55      0.55      3910

    accuracy                           0.42     20000
   macro avg       0.42      0.42      0.41     20000
weighted avg       0.41      0.42      0.41     20000

svm report:
              precision    recall  f1-score   support

           1       0.57      0.65      0.61      4081
           2       0.40      0.34      0.37      4029
           3       0.40      0.35      0.37      3965
           4       0.46      0.42      0.44      4015
           5       0.61      0.74      0.67      3910

    accuracy                           0.50     20000
   macro avg       0.49      0.50      0.49     20000
weighted avg       0.49      0.50      0.49     20000

logistic regression report:
              precision    recall  f1-score   support

           1       0.59      0.64      0.62      4081
           2       0.42      0.38      0.40      4029
           3       0.42      0.40      0.41      3965
           4       0.49      0.46      0.48      4015
           5       0.65      0.72      0.68      3910

    accuracy                           0.52     20000
   macro avg       0.51      0.52      0.52     20000
```

```
weighted avg       0.51      0.52      0.52     20000

naive bayes model report:
              precision    recall  f1-score   support

           1       0.60      0.59      0.60      4081
           2       0.40      0.38      0.39      4029
           3       0.39      0.40      0.40      3965
           4       0.46      0.45      0.46      4015
           5       0.64      0.68      0.66      3910

    accuracy                           0.50     20000
   macro avg       0.50      0.50      0.50     20000
weighted avg       0.50      0.50      0.50     20000
```

Extract the required fields from each report

In [21]:

```
def getreportvalues(rep):
    rep=rep.split("\n")
    rep=[i.split(" ") for i in rep]
    fin_rep=[]
    for i in rep:
        if i!=[]:
            temp=[]
            for j in i:
                if j!='':
                    temp.append(j)
            if temp!=[]:
                fin_rep.append(temp)
    return fin_rep
```

In [22]:

```
##perceptron report
rep=getreportvalues(percept)
for i in range(1,6):
    s=",".join(rep[i][1:4])
    print(s)
print(",".join(rep[-2][2:5]))
```

```
0.52,0.52,0.52
0.33,0.27,0.30
0.31,0.33,0.32
0.37,0.41,0.39
0.55,0.55,0.55
0.42,0.42,0.41
```

In [23]:
```
##svm report
rep=getreportvalues(svm_rep)
for i in range(1,6):
    s=",".join(rep[i][1:4])
    print(s)
print(",".join(rep[-2][2:5]))
```

```
0.57,0.65,0.61
0.40,0.34,0.37
0.40,0.35,0.37
0.46,0.42,0.44
0.61,0.74,0.67
0.49,0.50,0.49
```

In [24]:
```
##logistic regression report
rep=getreportvalues(log_reg_rep)
for i in range(1,6):
    s=",".join(rep[i][1:4])
    print(s)
print(",".join(rep[-2][2:5]))
```

```
0.59,0.64,0.62
0.42,0.38,0.40
0.42,0.40,0.41
0.49,0.46,0.48
0.65,0.72,0.68
0.51,0.52,0.52
```

In [25]:
```
## naive bayes report
rep=getreportvalues(nb_rep)
for i in range(1,6):
    s=",".join(rep[i][1:4])
    print(s)
print(",".join(rep[-2][2:5]))
```

```
0.60,0.59,0.60
0.40,0.38,0.39
0.39,0.40,0.40
0.46,0.45,0.46
0.64,0.68,0.66
0.50,0.50,0.50
```