CSCI544_HW1

- 09/08/2022

In [1]:
```
#version python3.9.2
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /home/chingyen_peng/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_Jewelry_v1_00.tsv.gz
```

Read Data

In [3]:
```
df = pd.read_csv('amazon_reviews_us_Jewelry_v1_00.tsv', sep='\t', on_bad_lines='skip')
```

```
/tmp/ipykernel_119695/2867304037.py:1: DtypeWarning: Columns (7) have mixed types. Specify dtype option on im
port or set low_memory=False.
  df = pd.read_csv('amazon_reviews_us_Jewelry_v1_00.tsv', sep='\t', on_bad_lines='skip')
```

Keep Reviews and Ratings

In [4]:
```
data = df[['star_rating','review_body']]
df = df.dropna()
```

We select 20000 reviews randomly from each rating class.

- combine all the selected reviews together
- reset the index
- split the dataset into 80% training dataset and 20% testing dataset

In [5]:
```
s1=df[df.star_rating == 1]
s2=df[df.star_rating == 2]
s3=df[df.star_rating == 3]
s4=df[df.star_rating == 4]
s5=df[df.star_rating == 5]
#print(len(s1),",",len(s2),",",len(s3),",",len(s4),",",len(s5))
s1 = s1.sample(n = 20000, random_state = None)
s2 = s2.sample(n = 20000, random_state = None)
s3 = s3.sample(n = 20000, random_state = None)
s4 = s4.sample(n = 20000, random_state = None)
s5 = s5.sample(n = 20000, random_state = None)
#print(len(s1),",",len(s2),",",len(s3),",",len(s4),",",len(s5))
dataset = pd.concat([s1, s2, s3, s4, s5])
dataset = dataset.reset_index(drop = True)
train = dataset.sample(frac = 0.8, random_state = None)
test = dataset.drop(train.index)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
#print(len(train),",",len(test))
```

Data Cleaning

- calculate the average length of the reviews in terms of character length in the dataset before cleaning.
- convert the all reviews into the lower case.

In [6]:
```
ave1 = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/100000
train['review_body'] = train['review_body'].str.lower()
test['review_body'] = test['review_body'].str.lower()
```

- remove the HTML and URLs from the reviews

In [7]:
```
train['review_body'] = train['review_body'].map(lambda x: re.sub(re.compile(r'[http|https]*://[a-zA-Z0-9.?/&=
test['review_body'] = test['review_body'].map(lambda x: re.sub(re.compile(r'[http|https]*://[a-zA-Z0-9.?/&=:]
train['review_body'] = train['review_body'].map(lambda x: BeautifulSoup(x,"html.parser").get_text())
test['review_body'] = test['review_body'].map(lambda x: BeautifulSoup(x,"html.parser").get_text())
```

```
/usr/local/lib/python3.9/dist-packages/bs4/__init__.py:435: MarkupResemblesLocatorWarning: The input looks mo
re like a filename than markup. You may want to open this file and pass the filehandle into Beautiful Soup.
  warnings.warn(
```

- remove non-alphabetical characters

In [8]:
```
train['review_body'] = train['review_body'].map(lambda x: re.sub("[^a-zA-Z]+", " ", x))
test['review_body'] = test['review_body'].map(lambda x: re.sub("[^a-zA-Z]+", " ", x))
```

- remove extra spaces

In [9]:
```
train['review_body'] = train['review_body'].map(lambda x: re.sub(" +", " ", x))
test['review_body'] = test['review_body'].map(lambda x: re.sub(" +", " ", x))
```

- perform contractions on the reviews
- calculate the average length of the reviews in terms of character length in the dataset after cleaning
- print these two average length

In [10]:
```
import contractions

def contractionFunc(s):
    s = contractions.fix(s)
    s = re.sub("[^a-zA-Z]+", " ", s)
    return s
train['review_body'] = train['review_body'].map(lambda x: contractionFunc(x))
test['review_body'] = test['review_body'].map(lambda x: contractionFunc(x))

ave2 = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/100000
print(ave1,',',ave2)
```

```
188.96851 , 182.21615
```

Pre-processing

remove the stop words

- use the stopwords resource in nltk, to filter the stopwords in reviews.

In [11]:
```
from nltk.corpus import stopwords
import nltk
nltk.download('stopwords')

stopwords = set(stopwords.words('english'))
def removeStopwords(s):
    items = s.split()
    string = ""
    for item in items:
        if item not in stopwords:
            string = string + item + " "
    return string.split()
train['review_body'] = train['review_body'].map(lambda x: removeStopwords(x))
test['review_body'] = test['review_body'].map(lambda x: removeStopwords(x))
#print(train.review_body)
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /home/chingyen_peng/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

perform lemmatization

- use omw-1.4 resource in nltk to lemmatize words
- calculate the average length of the reviews in terms of character length in the dataset after preprocessing
- print the two average length befero and after preprocessing

In [12]:
```
from nltk.stem import WordNetLemmatizer
import nltk
nltk.download('omw-1.4')

wordnet_lemmatizer = WordNetLemmatizer()
def lemmatization(s):
```

```
    lemmatize_tokens = [wordnet_lemmatizer.lemmatize(word) for word in s]
    s = ' '.join(lemmatize_tokens)
    return s
train['review_body'] = train['review_body'].map(lambda x: lemmatization(x))
test['review_body'] = test['review_body'].map(lambda x: lemmatization(x))
#print(train.review_body)

ave3 = (sum(train.review_body.str.len()) + sum(test.review_body.str.len()))/100000
print(ave2,',',ave3)
```

```
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /home/chingyen_peng/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
182.21615 , 107.43709
```

TF-IDF Feature Extraction

- extract TF-IDF feature of reviews of training and testing dataset
- built four set for training and testing-- X_train, Y_train, X_test, Y_test

In [13]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vect = TfidfVectorizer(use_idf=True, smooth_idf=True, norm=None, min_df = 0.001)
X_train = tfidf_vect.fit_transform(train['review_body'])
X_train = pd.DataFrame(X_train.toarray(), columns = tfidf_vect.get_feature_names_out())
X_test = tfidf_vect.transform(test['review_body'])
X_test = pd.DataFrame(X_test.toarray(), columns = tfidf_vect.get_feature_names_out())
Y_train = train['star_rating']
Y_test = test['star_rating']
#print(X_train)
#print(Y_train)
```

Perceptron

In [14]:
```
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score,f1_score,recall_score
from sklearn.linear_model import Perceptron

perceptron = Perceptron(max_iter = 1000, tol = 0, random_state = 200, eta0 = 0.01)
```

```
perceptron.fit(X_train, Y_train.astype('int'))
Y_test_pred = perceptron.predict(X_test)
#print(classification_report(Y_test.astype('int'), Y_test_pred))

score_p = precision_score(Y_test.astype('int'), Y_test_pred, average = None)
score_r = recall_score(Y_test.astype('int'), Y_test_pred, average = None)
score_f = f1_score(Y_test.astype('int'), Y_test_pred, average=None)
print("%.2f" % score_p[0], ", %.2f" % score_r[0], ", %.2f" % score_f[0])
print("%.2f" % score_p[1], ", %.2f" % score_r[1], ", %.2f" % score_f[1])
print("%.2f" % score_p[2], ", %.2f" % score_r[2], ", %.2f" % score_f[2])
print("%.2f" % score_p[3], ", %.2f" % score_r[3], ", %.2f" % score_f[3])
print("%.2f" % score_p[4], ", %.2f" % score_r[4], ", %.2f" % score_f[4])
average_p = precision_score(Y_test.astype('int'), Y_test_pred, average = 'macro')
average_r = recall_score(Y_test.astype('int'), Y_test_pred, average='macro')
average_f = f1_score(Y_test.astype('int'), Y_test_pred, average='macro')
print("%.2f" % average_p, ", %.2f" % average_r, ", %.2f" % average_f)
```

```
0.51 , 0.50 , 0.50
0.32 , 0.34 , 0.33
0.31 , 0.25 , 0.28
0.38 , 0.40 , 0.39
0.55 , 0.60 , 0.58
0.42 , 0.42 , 0.42
```

SVM

In [17]:

```
from sklearn.svm import SVC, LinearSVC

svc = SVC(max_iter = 5000)
model_svc = svc.fit(X_train, Y_train.astype('int'))
Y_test_svm = model_svc.predict(X_test)
#print(classification_report(Y_test.astype('int'), Y_test_svm))

score_p = precision_score(Y_test.astype('int'), Y_test_svm, average = None)
score_r = recall_score(Y_test.astype('int'), Y_test_svm, average = None)
score_f = f1_score(Y_test.astype('int'), Y_test_svm, average=None)
print("%.2f" % score_p[0], ", %.2f" % score_r[0], ", %.2f" % score_f[0])
print("%.2f" % score_p[1], ", %.2f" % score_r[1], ", %.2f" % score_f[1])
print("%.2f" % score_p[2], ", %.2f" % score_r[2], ", %.2f" % score_f[2])
print("%.2f" % score_p[3], ", %.2f" % score_r[3], ", %.2f" % score_f[3])
print("%.2f" % score_p[4], ", %.2f" % score_r[4], ", %.2f" % score_f[4])
average_p = precision_score(Y_test.astype('int'), Y_test_svm, average = 'macro')
```

```
average_r = recall_score(Y_test.astype('int'), Y_test_svm, average='macro')
average_f = f1_score(Y_test.astype('int'), Y_test_svm, average='macro')
print("%.2f" % average_p, ", %.2f" % average_r, ", %.2f" % average_f)
```

```
/home/chingyen_peng/.local/lib/python3.9/site-packages/sklearn/svm/_base.py:301: ConvergenceWarning: Solver t
erminated early (max_iter=5000).  Consider pre-processing your data with StandardScaler or MinMaxScaler.
  warnings.warn(
0.49 , 0.68 , 0.57
0.35 , 0.28 , 0.31
0.36 , 0.30 , 0.33
0.37 , 0.24 , 0.29
0.52 , 0.69 , 0.59
0.42 , 0.44 , 0.42
```

Logistic Regression

In [15]:
```
from sklearn.linear_model import LogisticRegression

LR = LogisticRegression(solver='liblinear',random_state=0, C=5, penalty='l2',max_iter=1000)
LR.fit(X_train, Y_train.astype('int'))
Y_test_LR = LR.predict(X_test)
#print(classification_report(Y_test.astype('int'), Y_test_LR))

score_p = precision_score(Y_test.astype('int'), Y_test_LR, average = None)
score_r = recall_score(Y_test.astype('int'), Y_test_LR, average = None)
score_f = f1_score(Y_test.astype('int'), Y_test_LR, average=None)
print("%.2f" % score_p[0], ", %.2f" % score_r[0], ", %.2f" % score_f[0])
print("%.2f" % score_p[1], ", %.2f" % score_r[1], ", %.2f" % score_f[1])
print("%.2f" % score_p[2], ", %.2f" % score_r[2], ", %.2f" % score_f[2])
print("%.2f" % score_p[3], ", %.2f" % score_r[3], ", %.2f" % score_f[3])
print("%.2f" % score_p[4], ", %.2f" % score_r[4], ", %.2f" % score_f[4])
average_p = precision_score(Y_test.astype('int'), Y_test_LR, average = 'macro')
average_r = recall_score(Y_test.astype('int'), Y_test_LR, average='macro')
average_f = f1_score(Y_test.astype('int'), Y_test_LR, average='macro')
print("%.2f" % average_p, ", %.2f" % average_r, ", %.2f" % average_f)
```

```
0.55 , 0.67 , 0.60
0.42 , 0.35 , 0.38
0.42 , 0.34 , 0.38
0.46 , 0.42 , 0.44
0.60 , 0.74 , 0.66
0.49 , 0.50 , 0.49
```

Multinomial Naive Bayes

In [16]:
```
from sklearn.naive_bayes import MultinomialNB

MNB = MultinomialNB()
MNB.fit(X_train, Y_train.astype('int'))
Y_test_MNB = MNB.predict(X_test)
#print(classification_report(Y_test.astype('int'), Y_test_MNB))

score_p = precision_score(Y_test.astype('int'), Y_test_MNB, average = None)
score_r = recall_score(Y_test.astype('int'), Y_test_MNB, average = None)
score_f = f1_score(Y_test.astype('int'), Y_test_MNB, average=None)
print("%.2f" % score_p[0], ", %.2f" % score_r[0], ", %.2f" % score_f[0])
print("%.2f" % score_p[1], ", %.2f" % score_r[1], ", %.2f" % score_f[1])
print("%.2f" % score_p[2], ", %.2f" % score_r[2], ", %.2f" % score_f[2])
print("%.2f" % score_p[3], ", %.2f" % score_r[3], ", %.2f" % score_f[3])
print("%.2f" % score_p[4], ", %.2f" % score_r[4], ", %.2f" % score_f[4])
average_p = precision_score(Y_test.astype('int'), Y_test_MNB, average = 'macro')
average_r = recall_score(Y_test.astype('int'), Y_test_MNB, average='macro')
average_f = f1_score(Y_test.astype('int'), Y_test_MNB, average='macro')
print("%.2f" % average_p, ", %.2f" % average_r, ", %.2f" % average_f)
```

```
0.56 , 0.62 , 0.59
0.42 , 0.34 , 0.38
0.40 , 0.37 , 0.38
0.43 , 0.40 , 0.42
0.60 , 0.73 , 0.66
0.48 , 0.49 , 0.48
```

In [ ]:
```

```