NLP
Report HW-1
Submitted by: Kartikeya Syal (5731454946)

1. I first read the tsv file skipping the bad lines. I have concatenated review_headline and review body into review_body column, as this was giving a better performance. I selected only 2 columns(review_body and star_rating) required for the hw. Then i divided the data into classes by adding another column which will put each review in buckets (class1, class2 and class3).
   I appended the data for these 3 classes into single dataframe after selecting random 20000 values of each class into a single dataframe named t4.

Data Cleaning:

2. **The average length before data cleaning was 213.**

   a. I changed all reviews into lowercase.
   b. Removed all html tags and url from the column and removed extra spaces.
   c. I performed contractions on the review body first as this helped me in increasing the performance.
   d. I then removed all the non-alphabetical characters and applied few more data cleaning techniques. First was to remove repeated words and then remove repeated characters in a single word. (for example: aaaaaaweeesomeeeee -> awesome)

   The average length after data cleaning came down to 204.

Data Preprocessing:

3. **The average length before data cleaning was 204.**
   a. I remove stop words first using nltk.corpus import stopwords. Stop words included (I , how , am, we, etc..)
   b. I then performed lemmatization using nltk.stem import WordNetLemmatizer to get the root words from the big words.

   The average length after data cleaning came down to 128.

4. I performed **tf-idf feature extraction** to get the importance of each term. I have marked down the print statement for extracted words but with that result i got to know that importance of each word. It also helped me during the cleaning processes as earlier i was seeing few numeric data as well in tf-idf features which gave me an indication that i need to clean the data properly. I used n-gram range as well here which showed me the importance of collection of data. (for example: if a review contains word ‘bad’ but a collection of words like ‘not bad’ completely changes the meaning of review)

5. I split the data into training and test data which will help me to train the model. 80/20 was the split.

6. **77% average performance.** Finally I fitted my values in Perceptron model and found recall, precision and f1 values and the average across all the classes.

	precision	recall	f1-score
class_1	0.79,	0.74,	0.77
class_2	0.69,	0.74,	0.71
class_3	0.83,	0.82,	0.82
avg	0.77,	0.77,	0.77

7. **80% average performance.** I fitted my values in SVM model using linear SVC and found recall, precision and f1 values and the average across all the classes.

	precision	recall	f1-score
class_1	0.80,	0.82,	0.81
class_2	0.75,	0.73,	0.74
class_3	0.85,	0.86,	0.86
avg	0.80,	0.80,	0.80

8. **80% average performance.** I fitted my values in Logistic Regression model and found recall, precision and f1 values and the average across all the classes.

	precision	recall	f1-score
class_1	0.80,	0.82,	0.81
class_2	0.75,	0.73,	0.74
class_3	0.85,	0.86,	0.86
avg	0.80,	0.80,	0.80

9. **78% average performance.** I fitted my values in Naive Bayes model using MultinomialNB and found recall, precision and f1 values and the average across all the classes.

	precision	recall	f1-score
class_1	0.78,	0.81,	0.79
class_2	0.69,	0.78,	0.73

class_3	0.90,	0.75,	0.82
avg	0.79,	0.78,	0.78

I have not used any hyperparameter tuning such as grid search as the prof. mentioned that we just need to use the data cleaning and pre-processing techniques to reach the model precision values.

I have kept 12 lines comma separated precision, recall, f1-score and average and also classification matrix in the print statement for your reference. Both will give a better view of the model performance and hence I have put both.

In [5]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/kartikeyasyal/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

In [8]:
```
! pip install bs4 # in case you don't have it installed

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

```
Requirement already satisfied: bs4 in /Users/kartikeyasyal/opt/anaconda3/lib/python3.9/site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in /Users/kartikeyasyal/opt/anaconda3/lib/python3.9/site-packages (from
bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in /Users/kartikeyasyal/opt/anaconda3/lib/python3.9/site-packages (from
beautifulsoup4->bs4) (2.3.1)
```

Read Data

In [1]:
```
import pandas as pd
df = pd.read_table('amazon_reviews_us_Beauty_v1_00.tsv', on_bad_lines='skip')
df["review_body"] = df['review_body'].astype(str) + df["review_headline"].astype(str)
```

```
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/665461994.py:2: DtypeWarning: Columns (7) have mixed
types. Specify dtype option on import or set low_memory=False.
  df = pd.read_table('amazon_reviews_us_Beauty_v1_00.tsv', on_bad_lines='skip')
```

In [ ]:
```
```

Keep Reviews and Ratings

In [2]:
```
t = df[["star_rating","review_body"]]
```

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

In [87]:
```
conditions = [
    (t['star_rating'] == '1') | (t['star_rating'] == '2'),
    (t['star_rating'] == '3'),
    (t['star_rating'] == '4') | (t['star_rating'] == '5')
]
choices = ['class_1', 'class_2','class_3']
t['classes'] = np.select(conditions,choices, default=0)


t1 = t[t.classes == "class_1"].sample(n=20000)
t2 = t[t.classes == "class_2"].sample(n=20000)
t3 = t[t.classes == "class_3"].sample(n=20000)
```

```
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/3289960237.py:7: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning
-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-
a-copy)
  t['classes'] = np.select(conditions,choices, default=0)
```

In [88]:
```
t4 = pd.DataFrame(columns = ['star_rating', 'review_body', 'classes'])
t4 = t4.append(t1, ignore_index = True)
t4 = t4.append(t2, ignore_index = True)
t4 = t4.append(t3, ignore_index = True)
```

```
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/4103022133.py:2: FutureWarning: The frame.append met
hod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  t4 = t4.append(t1, ignore_index = True)
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/4103022133.py:3: FutureWarning: The frame.append met
hod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  t4 = t4.append(t2, ignore_index = True)
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/4103022133.py:4: FutureWarning: The frame.append met
hod is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
  t4 = t4.append(t3, ignore_index = True)
```

Data Cleaning

In [89]:
```
result1_3 = t4['review_body'].str.len().mean()
print("average length before data cleaning", result1_3)

t4['review_body'] = t4['review_body'].str.lower()
import re
t4['review_body'] = t4['review_body'].apply(lambda x: re.split('https:\/\/.*', str(x))[0])
t4['review_body'] = t4['review_body'].str.replace(r'<[^<>]*>', '', regex=True)
t4['review_body'].str.strip()

import contractions
t4["review_body"] = t4['review_body'].apply(lambda x: [contractions.fix(word) for word in x.split()])
t4['review_body'] = [','.join(map(str, l)) for l in t4['review_body']]
t4 = t4.replace(',',' ', regex=True)


t4['review_body'] = t4['review_body'].str.replace('[^a-zA-Z]', ' ', regex=True).str.strip()

min_threshold_rep = 1
t4['review_body']= t4['review_body'].str.replace(r'(\w)\1{%d,}'%(min_threshold_rep-1), r'\1')
t4["review_body"] = t4["review_body"].str.replace(r"\s+(.)\1+\b", "").str.strip()
t4['review_body'] = t4['review_body'].str.replace(r'\b(\w+)(\s+\1)+\b', r'\1')


result1_3 = t4['review_body'].str.len().mean()

print("average length after data cleaning", result1_3)
```

```
average length before data cleaning 213.87788333333333

/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/273994230.py:19: FutureWarning: The default value of
regex will change from True to False in a future version.
  t4['review_body']= t4['review_body'].str.replace(r'(\w)\1{%d,}'%(min_threshold_rep-1), r'\1')
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/273994230.py:20: FutureWarning: The default value of
regex will change from True to False in a future version.
  t4["review_body"] = t4["review_body"].str.replace(r"\s+(.)\1+\b", "").str.strip()
/var/folders/0d/ptx8nwr965v0v978_xzgnqj00000gn/T/ipykernel_47211/273994230.py:21: FutureWarning: The default value of
regex will change from True to False in a future version.
  t4['review_body'] = t4['review_body'].str.replace(r'\b(\w+)(\s+\1)+\b', r'\1')

average length after data cleaning 204.80065
```

Pre-processing

remove the stop words

In [90]:
```
result3 = t4["review_body"].apply(len).mean()
print("Average length before data pre-processing", result3)


from nltk.corpus import stopwords
nltk.download('stopwords')

stop = stopwords.words('english')
t4['review_body'] = t4['review_body'].apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
```

```
Average length before data pre-processing 204.80065

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

perform lemmatization

In [91]:
```
import nltk
nltk.download('omw-1.4')
from nltk.stem import WordNetLemmatizer
lmtzr = WordNetLemmatizer()
t4['review_body'] = t4['review_body'].apply(lambda word: lmtzr.lemmatize(word, pos='v'))

result3 = t4["review_body"].apply(len).mean()
print("Average length after data pre-processing", result3)
```

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

Average length after data pre-processing 128.82946666666666
```

TF-IDF Feature Extraction

In [92]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import classification_report

tfidf = TfidfVectorizer(max_features = None, ngram_range = (1,5), analyzer = 'char')

X1 = tfidf.fit_transform(t4['review_body'])
y1 = t4['classes']

corpus = []
for x in t4["review_body"]:
    corpus.append(x)

cv = TfidfVectorizer(ngram_range = (1,1))
x = cv.fit_transform(corpus)
#print(cv.vocabulary_)
```

In [93]:
```
all_feature_names = cv.get_feature_names_out()
for word in all_feature_names:
    indx = cv.vocabulary_.get(word)
    #print(f"{word} {cv.idf_[indx]}")
```

In [94]:
```
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size = 0.2, random_state=0)
```

Perceptron

In [95]:
```
from sklearn.datasets import load_digits
from sklearn.linear_model import Perceptron
from sklearn.metrics import f1_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score


clf = Perceptron()
clf.fit(X1_train, y1_train)

print("\n\n  Perceptron model:")


y1_pred = clf.predict(X1_test)
print("          [class1              class2              class3]               average")
print("precision", precision_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  precision_score(y1_test,
print("f1       ", f1_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  f1_score(y1_test, y1_pred, avera
print("recall   " , recall_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  recall_score(y1_test, y1_pr

print("\n \n")
#print(f1_score(y1_test, y1_pred))
#print 'Recall:', recall_score(y_test, prediction)
#print 'Precision:', precision_score(y_test, prediction)

print(classification_report(y1_test, y1_pred))
```

```
          [class1              class2              class3]               average
precision [0.7933065595716198, 0.6911322065292892, 0.8256462240243284] , 0.77
f1        [0.7664252457320229, 0.7145421903052065, 0.8234550739289775] , 0.77
recall    [0.7413059794846134, 0.7395936570862239, 0.8212755230652886] , 0.77


              precision    recall  f1-score   support

     class_1       0.79      0.74      0.77      3997
     class_2       0.69      0.74      0.71      4036
     class_3       0.83      0.82      0.82      3967

    accuracy                           0.77     12000
   macro avg       0.77      0.77      0.77     12000
weighted avg       0.77      0.77      0.77     12000
```

SVM

In [96]:
```
clf = LinearSVC()

clf.fit(X1_train, y1_train)
y1_pred = clf.predict(X1_test)

print("\n Logistic Regression:")

print("          [class1              class2              class3]               average")
print("precision", precision_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  precision_score(y1_test,
print("f1       ", f1_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  f1_score(y1_test, y1_pred, avera
print("recall   " , recall_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  recall_score(y1_test, y1_pr

print("\n \n")
print(classification_report(y1_test, y1_pred))
```

```
          [class1              class2              class3]               average
precision [0.7957540263543191, 0.749234693877551, 0.8543445504771472] , 0.80
f1        [0.8056825200741197, 0.7383107088989442, 0.855956724116241] , 0.80
recall    [0.8158618964223168, 0.7277006937561943, 0.8575749936980086] , 0.80


              precision    recall  f1-score   support

     class_1       0.80      0.82      0.81      3997
     class_2       0.75      0.73      0.74      4036
     class_3       0.85      0.86      0.86      3967

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

Logistic Regression

In [97]:
```
om sklearn.linear_model import LogisticRegression

greg =  LogisticRegression()

greg.fit(X1_train,y1_train)
_pred = clf.predict(X1_test)
int("\n Logistic Regression:")
int("          [class1              class2              class3]               average")
int("precision", precision_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  precision_score(y1_test, y1_
int("f1       ", f1_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  f1_score(y1_test, y1_pred, average
int("recall   " , recall_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  recall_score(y1_test, y1_pred,

int("\n \n")
int(classification_report(y1_test, y1_pred))
```

```
/Users/kartikeyasyal/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: ConvergenceWarn
ing: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scikit-learn.org/stable/modules/preprocessin
g.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression (https://scikit-learn.org/stable/mo
dules/linear_model.html#logistic-regression)
  n_iter_i = _check_optimize_result(

          [class1              class2              class3]               average
precision [0.7957540263543191, 0.749234693877551, 0.8543445504771472] , 0.80
f1        [0.8056825200741197, 0.7383107088989442, 0.855956724116241] , 0.80
recall    [0.8158618964223168, 0.7277006937561943, 0.8575749936980086] , 0.80


              precision    recall  f1-score   support

     class_1       0.80      0.82      0.81      3997
     class_2       0.75      0.73      0.74      4036
     class_3       0.85      0.86      0.86      3967

    accuracy                           0.80     12000
   macro avg       0.80      0.80      0.80     12000
weighted avg       0.80      0.80      0.80     12000
```

Naive Bayes

In [98]:
```
sklearn.datasets import load_iris
rt numpy as np
sklearn.naive_bayes import MultinomialNB
sklearn.linear_model import LinearRegression

= MultinomialNB()

fit(X1_train, y1_train)

red = gnb.predict(X1_test)

t("\n Naive Bayes:")

t("          [class1              class2              class3]               average")
t("precision", precision_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  precision_score(y1_test, y1_pr
t("f1       ", f1_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  f1_score(y1_test, y1_pred, average =
t("recall   " , recall_score(y1_test, y1_pred,average = None).tolist(), "," , "%.2f" %  recall_score(y1_test, y1_pred, a

t("\n \n")
t(classification_report(y1_test, y1_pred))
```

```
          [class1              class2              class3]               average
precision [0.7790950883135737, 0.689094107299912, 0.8975595058752637] , 0.79
f1        [0.7921279212792128, 0.7301957129543337, 0.8177326379357672] , 0.78
recall    [0.8056042031523643, 0.7765113974231913, 0.7509452987143937] , 0.78


              precision    recall  f1-score   support

     class_1       0.78      0.81      0.79      3997
     class_2       0.69      0.78      0.73      4036
     class_3       0.90      0.75      0.82      3967

    accuracy                           0.78     12000
   macro avg       0.79      0.78      0.78     12000
weighted avg       0.79      0.78      0.78     12000
```

In [ ]: