Name: Dipali Chandrakant Telavane
USC ID: 1487027774

HW1 Report

1. External libraries used
   Following external libraries were used in the homework:
   bs4, contractions, scikit-learn

   Command to install these libraries:
   !pip install bs4
   !pip install contractions
   !pip install scikit-learn

   nltk packages downloaded:
   nltk.download('wordnet')
   nltk.download('stopwords')

2. Python version
   - Python 3.11.1

3. Dataset input
   - First unzipped the dataset in current location and imported it in dataframe.
   - Removed the rows in dataset giving error using *on_bad_lines='skip'*
   - Kept only review and rating columns and removed other columns. Also added another column named 'class' to label data in class 1, 2, 3.
   - Selected 20000 samples from each class.
   - Did not split the data into training and testing data.

4. Data Cleaning
   Performed the following steps of cleaning as part of data cleaning:
   - Data changed to lower case
   - Removed html tags using external library BeautifulSoup
   - Removed non-alphabetical words using Regex
   - Removed white spaces
   - Expanded contractions using external contractions library. For ex. Can't -> Can not

   The average review character length before and after cleaning: 273.44127010584214, 263.40793333333335

5. Data Pre-Processing
   - Removed the stop words and lemmatized the reviews in this step.
   - Used the English language stop words provided in *nltk* library.
   - Lemmatized verbs, noun, adjectives, adverbs, satellite adjectives sequentially from reviews data.

- Used pos attribute in lemmatize function to identify which part of the speech to be lemmatized.
- 'n' value for pos indicates lemmatizing of nouns, 'v' indicates verbs, 'a' indicates adjectives, 'r' indicates adverbs and 's' indicates satellite adjectives.
- Average review character length before and after pre-processing: 263.40793333333335 , 152.23136666666667

6. TF_IDF Feature extraction
   - Used the inbuilt *TfidfVectorizer* from *sklearn* library for feature extraction
   - Created a vector matrix of features and their TF-IDF value

7. Split the data
   - Split the data into training data and testing data with 80:20 ratio using sklearn inbuilt implementation.
   - Used the stratify parameter while splitting, to maintain enough data of every class.
   - Mentioned random_state = 42, to ensure that same data is used in splitting in every run.

8. Perceptron model
   - Used sklearn in-built implementation of Perceptron model.
   - Trained the model using training data and tested it using test data.
   - The average precision for perceptron model was around 0.59, recall was 0.59 and f1-score was around 0.59.
   - Class wise precision, recall and f-1 score was as follows:

	Precision	Recall	f-1 score
class1:	0.6009032564772997	0.632	0.6160594614353601
class2:	0.5062782521346058	0.504	0.5051365572538211
class3:	0.6688533193387562	0.63725	0.6526693125080015
average:	0.5920116093168872	0.5910833333333333	0.5912884437323942

9. SVM model
   - Used sklearn in-built implementation of SVM model.
   - Used LinearSVC instead of SVM since SVM implementation was using Kernel function and was taking a lot of time to train.
   - Trained the model using training data and tested it using test data.
   - The average precision for perceptron model was around 0.65, recall was 0.65 and f1-score was around 0.65.
   - Class wise precision, recall and f-1 score was as follows:

Name: Dipali Chandrakant Telavane
USC ID: 1487027774

	Precision	Recall	f-1 score
class1:	0.6601401982112642	0.68275	0.67125476219737
class2:	0.5764705882352941	0.539	0.5571059431524549
class3:	0.7215619694397284	0.74375	0.7324879970454267
average:	0.6527242519620956	0.6551666666666667	0.6536162341317505

10. Logistic Regression model
    - Used sklearn in-built implementation of Logistic Regression model.
    - Trained the model using training data and tested it using test data.
    - The average precision for perceptron model was around 0.67, recall was 0.67 and f1-score was around 0.67.
    - Class wise precision, recall and f-1 score was as follows:

	Precision	Recall	f-1 score
class1:	0.6754726126999515	0.69675	0.6859463450652227
class2:	0.5952929137886928	0.58175	0.5884435453281072
class3:	0.7465321563682219	0.74	0.7432517263025737
average:	0.6724325609522888	0.6728333333333333	0.6725472055653011

11. Multinomial Naïve Bayes model
    - Used sklearn in-built implementation of Multinomial Naïve Bayes model.
    - Trained the model using training data and tested it using test data.
    - The average precision for perceptron model was around 0.66, recall was 0.65 and f1-score was around 0.65.
    - Class wise precision, recall and f-1 score was as follows:

	Precision	Recall	f-1 score
class1:	0.6812717071867486	0.6375	0.6586594343277798
class2:	0.5502413339183853	0.627	0.5861182519280206
class3:	0.7512841308461746	0.69475	0.7219119366151447
average:	0.6609323906504362	0.6530833333333333	0.6555632076236484

Name: Dipali Chandrakant Telavane
USC ID: 1487027774

12. Observations
    - Logistic regression gave highest accuracy out of all models.
    - Class 2 had the lowest prediction accuracy overall as the data for it was comparatively lesser than other classes.

13. References

- https://stackoverflow.com/questions/18039057/python-pandas-error-tokenizing-data
- https://sparkbyexamples.com/pandas/pandas-apply-with-lambda-examples/#:~:text=Apply%20Lambda%20Expression%20to%20Single,x%3Ax%2D2)%20.
- https://stackoverflow.com/questions/67174746/sklearn-take-only-few-records-from-each-target-class
- https://stackoverflow.com/questions/39175963/sampling-n-2000-from-a-dask-dataframe-of-len-18000-generates-error-cannot-take
- https://www.reddit.com/r/learnpython/comments/37bdv3/stringlower_versus_stringlower/
- https://www.w3resource.com/python-exercises/pandas/string/python-pandas-string-exercise-41.php
- https://stackoverflow.com/questions/51994254/removing-url-from-a-column-in-pandas-dataframe
- https://stackoverflow.com/questions/19790188/expanding-english-language-contractions-in-python
- https://stackoverflow.com/questions/52979330/how-to-get-the-average-length-of-the-strings-in-each-column-in-csv
- https://stackoverflow.com/questions/51460881/pandas-typeerror-object-of-type-float-has-no-len
- https://stackoverflow.com/questions/29523254/python-remove-stop-words-from-pandas-dataframe
- https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
- https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

Name: Dipali Chandrakant Telavane
USC ID: 1487027774

- https://stackoverflow.com/questions/37593293/how-to-get-tfidf-with-pandas-dataframe
- https://www.geeksforgeeks.org/sklearn-feature-extraction-with-tf-idf/
- https://www.quantstart.com/articles/training-the-perceptron-with-scikit-learn-and-tensorflow/
- https://python-course.eu/machine-learning/perceptron-class-in-sklearn.php
- https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/
- https://stackoverflow.com/questions/27912872/what-is-the-difference-between-svc-and-svm-in-scikit-learn
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://www.tutorialspoint.com/scikit_learn/scikit_learn_multinomial_naive_bayes.htm
- https://www.machinelearningplus.com/nlp/lemmatization-examples-python/
- https://www.nltk.org/_modules/nltk/stem/wordnet.html

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

#!pip install bs4
#!pip install contractions
#!pip install scikit-learn

import contractions
from bs4 import BeautifulSoup

#remove warnings in output
import warnings
warnings.filterwarnings('ignore')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\dipal\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_Beauty_v1_00.tsv.gz
```

Read Data

In [3]:
```
#skipping rows in dataset which give error
#reference: https://stackoverflow.com/questions/18039057/python-pandas-error-tokenizing-data
df = pd.read_table("amazon_reviews_us_Beauty_v1_00.tsv", on_bad_lines='skip')
```

Keep Reviews and Ratings

In [4]:
```
df = df[["star_rating", "review_body"]]
#df.head()
```

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

In [5]:
```
#adding new column named "class" to define class 1,2 and 3
#reference: https://sparkbyexamples.com/pandas/pandas-apply-with-lambda-examples/#:~:text=Apply%20Lambda%20Expression%20to%20
df["class"] = df["star_rating"].apply(lambda x : 3 if str(x) > '3' else 2 if str(x) == '3' else 1)

#select 20000 reviews from each class
#reference: https://stackoverflow.com/questions/67174746/sklearn-take-only-few-records-from-each-target-class
df = df.groupby('class').sample(n=20000, replace=True)
```

Data Cleaning

Pre-processing

In [6]:
```
df_train = df[["review_body", "class"]]

charlenpre = df_train['review_body'].str.len().mean()

#review to lower case
df_train['review_body'] = df_train['review_body'].apply(lambda x : str(x).lower())

#remove html tags
#reference: https://stackoverflow.com/questions/753052/strip-html-from-strings-in-python
df_train['review_body'] = df_train['review_body'].apply(lambda x: BeautifulSoup(str(x)).get_text())

#remove url
#reference: https://stackoverflow.com/questions/51994254/removing-url-from-a-column-in-pandas-dataframe
df_train['review_body'] = df_train['review_body'].apply(lambda x: re.split('https:\/\/.*', str(x))[0])

#remove non-alphabetical words
df_train['review_body'] = df_train['review_body'].replace('[^a-zA-Z ]', '', regex=True)

#remove extra spaces
df_train['review_body'] = df_train['review_body'].str.strip()

#perform contractions
#reference: https://stackoverflow.com/questions/19790188/expanding-english-language-contractions-in-python
df_train['review_body'] = df_train['review_body'].apply(lambda x: contractions.fix(str(x)))

charlenpost = df_train['review_body'].str.len().mean()

print("Average review character length before and after cleaning: ", charlenpre, ",", charlenpost)
```

```
Average review character length before and after cleaning:  273.44127010584214 , 263.40793333333335
```

remove the stop words

In [7]:
```
from nltk.corpus import stopwords

nltk.download('stopwords')

charlenpre = df_train['review_body'].str.len().mean()

stopwords = stopwords.words('english')


#remove the words which are present in stopwords
#reference: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([word for word in x.split()
                                                                            if word not in (stopwords)]))
```

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

perform lemmatization

In [8]:
```
from nltk.stem import WordNetLemmatizer
from nltk.corpus import wordnet

#perform lemmatization of words
#reference: 1. https://www.geeksforgeeks.org/python-lemmatization-with-nltk/
#           2. https://www.nltk.org/_modules/nltk/stem/wordnet.html
lemmatizer = WordNetLemmatizer()

#verbs
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, pos="v")
                                                                            for word in x.split()]))

#noun
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, pos="n")
                                                                            for word in x.split()]))

#adjectives
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, pos="a")
                                                                            for word in x.split()]))

#adverbs
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, pos="r")
                                                                            for word in x.split()]))

#satellite adjectives
df_train['review_body'] = df_train['review_body'].apply(lambda x: ' '.join([lemmatizer.lemmatize(word, pos="s")
                                                                            for word in x.split()]))

charlenpost = df_train['review_body'].str.len().mean()

print("Average review character length before and after pre-processing: ",charlenpre, ", ", charlenpost)

#df_train.head(50)
```

```
Average review character length before and after pre-processing:  263.40793333333335 ,  152.23136666666667
```

TF-IDF Feature Extraction

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

#tf-idf feature extraction of input
#reference: https://stackoverflow.com/questions/37593293/how-to-get-tfidf-with-pandas-dataframe
vectorizer = TfidfVectorizer()
vector = vectorizer.fit_transform(df_train['review_body'])

#vector
```

Perceptron

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from sklearn.metrics import f1_score, precision_score, recall_score
from sklearn.metrics import classification_report

#split data into training and test
#reference: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html
Xtrain, Xtest, Ytrain, Ytest = train_test_split(vector, df_train['class'], stratify=df_train['class'],
                                                test_size=0.2, random_state=42)

#Perceptron model training
#reference: https://python-course.eu/machine-learning/perceptron-class-in-sklearn.php
model_p = Perceptron(random_state=42)
model_p.fit(Xtrain, Ytrain)

#Testing the model
Ypred = model_p.predict(Xtest)

precision_score_p = precision_score(Ytest, Ypred, average=None)
recall_score_p = recall_score(Ytest, Ypred, average=None)
f1_score_p = f1_score(Ytest, Ypred, average=None)

print("Perceptron model output:")
print("class1: ", precision_score_p[0], ", ", recall_score_p[0], ", ", f1_score_p[0])
print("class2: ", precision_score_p[1], ", ", recall_score_p[1], ", ", f1_score_p[1])
print("class3: ", precision_score_p[2], ", ", recall_score_p[2], ", ", f1_score_p[2])
print("average:", precision_score(Ytest, Ypred, average='weighted'),", ", recall_score(Ytest, Ypred, average='weighted'),
     ", ", f1_score(Ytest, Ypred, average='weighted'))


#print("training dataresults: ")
#print(classification_report(model_p.predict(Xtrain), Ytrain))

#print("testing dataresults: ")
#print(classification_report(Ypred, Ytest))
```

```
Perceptron model output:
class1:  0.6009032564772997 ,  0.632 ,  0.6160594614353601
class2:  0.5062782521346058 ,  0.504 ,  0.5051365572538211
class3:  0.6688533193387562 ,  0.63725 ,  0.6526693125080015
average: 0.5920116093168872 ,  0.5910833333333333 ,  0.5912884437323942
```

SVM

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

#Linear SVM model training
#reference: 1. https://stackabuse.com/implementing-svm-and-kernel-svm-with-pythons-scikit-learn/
#           2. https://stackoverflow.com/questions/27912872/what-is-the-difference-between-svc-and-svm-in-scikit-learn
model_s = LinearSVC(random_state=42)
model_s.fit(Xtrain, Ytrain)

#Testing the model
Ypred = model_s.predict(Xtest)

precision_score_s = precision_score(Ytest, Ypred, average=None)
recall_score_s = recall_score(Ytest, Ypred, average=None)
f1_score_s = f1_score(Ytest, Ypred, average=None)

print("SVM model output:")
print("class1: ", precision_score_s[0], ", ", recall_score_s[0], ", ", f1_score_s[0])
print("class2: ", precision_score_s[1], ", ", recall_score_s[1], ", ", f1_score_s[1])
print("class3: ", precision_score_s[2], ", ", recall_score_s[2], ", ", f1_score_s[2])
print("average:", precision_score(Ytest, Ypred, average='weighted'), ", ", recall_score(Ytest, Ypred, average='weighted'),
      ", ", f1_score(Ytest, Ypred, average='weighted'))


#print("training dataresults: ")
#print(classification_report(model_s.predict(Xtrain), Ytrain))

#print("testing dataresults: ")
#print(classification_report(Ypred, Ytest))
```

```
SVM model output:
class1:  0.6601401982112642 ,  0.68275 ,  0.67125476219737
class2:  0.5764705882352941 ,  0.539 ,  0.5571059431524549
class3:  0.7215619694397284 ,  0.74375 ,  0.7324879970454267
average: 0.6527242519620956 ,  0.6551666666666667 ,  0.6536162341317505
```

Logistic Regression

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

#Logistic Regression model training
#reference: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
model_l = LogisticRegression(random_state=42)
model_l.fit(Xtrain, Ytrain)

#Testing the model
Ypred = model_l.predict(Xtest)

precision_score_l = precision_score(Ytest, Ypred, average=None)
recall_score_l = recall_score(Ytest, Ypred, average=None)
f1_score_l = f1_score(Ytest, Ypred, average=None)

print("Logistic Regression model output:")
print("class1: ", precision_score_l[0], ", ", recall_score_l[0], ", ", f1_score_l[0])
print("class2: ", precision_score_l[1], ", ", recall_score_l[1], ", ", f1_score_l[1])
print("class3: ", precision_score_l[2], ", ", recall_score_l[2], ", ", f1_score_l[2])
print("average:", precision_score(Ytest, Ypred, average='weighted'),", ", recall_score(Ytest, Ypred, average='weighted'),
      ", ", f1_score(Ytest, Ypred, average='weighted'))


#print("training dataresults: ")
#print(classification_report(model_l.predict(Xtrain), Ytrain))

#print("testing dataresults: ")
#print(classification_report(Ypred, Ytest))
```

```
Logistic Regression model output:
class1:  0.6754726126999515 ,  0.69675 ,  0.6859463450652227
class2:  0.5952929137886928 ,  0.58175 ,  0.5884435453281072
class3:  0.7465321563682219 ,  0.74 ,  0.7432517263025737
average: 0.6724325609522888 ,  0.6728333333333333 ,  0.6725472055653011
```

Naive Bayes

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

model_n = MultinomialNB()
model_n.fit(Xtrain, Ytrain)

#Testing the model
Ypred = model_n.predict(Xtest)

precision_score_n = precision_score(Ytest, Ypred, average=None)
recall_score_n = recall_score(Ytest, Ypred, average=None)
f1_score_n = f1_score(Ytest, Ypred, average=None)

print("Multinomial Naive Bayes model output:")
print("class1: ", precision_score_n[0], ", ", recall_score_n[0], ", ", f1_score_n[0])
print("class2: ", precision_score_n[1], ", ", recall_score_n[1], ", ", f1_score_n[1])
print("class3: ", precision_score_n[2], ", ", recall_score_n[2], ", ", f1_score_n[2])
print("average:", precision_score(Ytest, Ypred, average='weighted'),", ", recall_score(Ytest, Ypred, average='weighted'),
      ", ", f1_score(Ytest, Ypred, average='weighted'))

#print("training dataresults: ")
#print(classification_report(model_n.predict(Xtrain), Ytrain))

#print("testing dataresults: ")
#print(classification_report(Ypred, Ytest))
```

```
Multinomial Naive Bayes model output:
class1:  0.6812717071867486 ,  0.6375 ,  0.6586594343277798
class2:  0.5502413339183853 ,  0.627 ,  0.5861182519280206
class3:  0.7512841308461746 ,  0.69475 ,  0.7219119366151447
average: 0.6609323906504362 ,  0.6530833333333333 ,  0.6555632076236484
```