In [3]:
```
## Importing and installing libraries

import numpy as np
import pandas as pd
import warnings
import re
import sys
import nltk
from bs4 import BeautifulSoup
from nltk.corpus import stopwords
import string
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import Perceptron
from scipy.sparse import hstack
from sklearn.metrics import classification_report
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from statistics import mean
from sklearn.svm import LinearSVC

warnings.filterwarnings('ignore')

!pip install contractions
import contractions

nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('omw-1.4')
```

```
Requirement already satisfied: contractions in /Users/apurvagupta/opt/anaconda3/lib/python3.9/site-packages (0.1.73)
Requirement already satisfied: textsearch>=0.0.21 in /Users/apurvagupta/opt/anaconda3/lib/python3.9/site-packages (from contractions) (0.0.24)
Requirement already satisfied: anyascii in /Users/apurvagupta/opt/anaconda3/lib/python3.9/site-packages (from textsearch>=0.0.21->contractions) (0.3.1)
Requirement already satisfied: pyahocorasick in /Users/apurvagupta/opt/anaconda3/lib/python3.9/site-packages (from textsearch>=0.0.21->contractions) (2.0.0)
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/apurvagupta/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/apurvagupta/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     /Users/apurvagupta/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to
[nltk_data]     /Users/apurvagupta/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
```

Out[3]: True

1.Data Preparation

Preparing the new dataframe by retrieving the two columns needed from the originial dataset

In [4]:
```
#fields required in the balanced dataframe from the original dataset
input_column=["review_body","star_rating"]

#reading the original dataset to filter the columns that are required
input_df =pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv',usecols=input_column,sep='\t',error_bad_lines=False)
```

In [5]:
```
#Creating 3 different classes to get 20000 data from each class to avoid computational burden

class_one_df =(input_df[(input_df['star_rating'] == 1) | (input_df['star_rating'] == 2) ]).sample(n=20000)
class_one_df['class']=1

class_two_df =(input_df[(input_df['star_rating'] == 3)]).sample(n=20000)
class_two_df['class']=2

class_three_df =(input_df[(input_df['star_rating'] == 4) | (input_df['star_rating'] == 5) ]).sample(n=20000)
class_three_df['class']=3


#Combining all the data received from each class into a single balanced dataframe

amazon_balanced_df = pd.concat([class_one_df, class_two_df, class_three_df])

#Resetting the index as we have retrieved different data according to the classes created.
#Therefore, we will have irregular or unsorted index keys.
#We will reset the index to the new and incremental values from 0

amazon_balanced_df = amazon_balanced_df.reset_index(drop=True)

# Created a new dataframe consisting of the two columns (star_rating and review_body)
#along with class one assigned to them on the basis of star_rating. We are also resetting the index
```

2.Data Cleaning

Creating a dataframe and fetching the the review length before cleaning

In [6]:
```
df=pd.DataFrame()

#Calculating the length of the review body before cleaning

df['pre_clean_review_len'] = amazon_balanced_df['review_body'].str.len()
```

Handling null values

In [7]:
```
#We are changing all null values to an empty string

amazon_balanced_df = amazon_balanced_df.fillna('')
```

Convert all reviews into lowercase

In [8]:
```
# Converting all review body into lowercase

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.lower()
```

Remove the HTML from the reviews

In [9]:
```
# Removing all the html tags from each review body

amazon_balanced_df['review_body']=amazon_balanced_df['review_body'].apply(lambda x : re.sub('<.*?>','',str(x)))
```

Remove the URLs from the reviews

In [10]:
```
# Removing all the URLs from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda y: re.split('https:\/\/.*', str(y))[0])
```

Remove non-alphabetical characters

In [11]:
```
# Removing all the non alphabetic chaarcters(symbols, numbers) from each review body

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda z: " ".join([re.sub('[^A-Za-z]+','', z) for z in nltk.word_tokenize(z)]))
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(lambda z: re.sub(' +', ' ', z))
```

Remove extra spaces

In [13]:
```
# Will remove leading and trailing spaces
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.strip()
```

Perform contractions on the review_body

In [14]:
```
## This will elongate the short form used in sentences like (I'll ---> I will)

amazon_balanced_df['without_contraction'] = amazon_balanced_df['review_body'].apply(lambda a: [contractions.fix(word) for word in a.split()])
amazon_balanced_df['review_body'] = [' '.join(map(str, x)) for x in amazon_balanced_df['without_contraction']]
```

Remove Punctuations

In [15]:
```
amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].str.replace(r'[^\w\s]+', '')
```

Printing the post and pre cleaned review length

In [16]:
```
print("Average length of reviews before and after data cleaning \n")

print(df['pre_clean_review_len'].mean())

df['post_clean_review_len'] = amazon_balanced_df['review_body'].str.len()

print(df['post_clean_review_len'].mean())
```

```
Average length of reviews before and after data cleaning

286.99408313610456
274.4493166666667
```

3. Preprocessing

Before Preprocessing review length

In [17]:
```
df['before_preprocess'] = amazon_balanced_df['review_body'].str.len()
```

Removing the stop words

In [18]:
```
# Removing all the stop words(the, an, in, etc.) from each review body

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

Lemmatization: grouping together different inflicted form of words

In [19]:
```
word_lem = WordNetLemmatizer()

amazon_balanced_df['review_body'] = amazon_balanced_df['review_body'].apply(word_lem.lemmatize)
```

Getting Before and After Preprocessing review length

In [20]:
```
print("Average of Review Body Length before and after data Preprocessing \n")

print(df['before_preprocess'].mean())

df['after_preprocess'] = amazon_balanced_df['review_body'].str.len()

print(df['after_preprocess'].mean())
```

```
Average of Review Body Length before and after data Preprocessing

274.4493166666667
170.34295
```

4. Feature Extraction

Split the data into 80% train and 20% test

In [21]:
```
X_train, X_test, y_train, y_test = train_test_split(amazon_balanced_df['review_body'], amazon_balanced_df['class'], test_size=0.20, random_state=20)
```

In [22]:
```
Tf_Id_Vector = TfidfVectorizer(
    ngram_range=(1, 2),
    analyzer='word',
    token_pattern=r'\w{1,}',
    strip_accents='unicode',
    max_features=10000,
    stop_words='english'
)
```

In [23]:
```
X_tf_id_train = Tf_Id_Vector.fit_transform(X_train)

X_tf_id_test = Tf_Id_Vector.transform(X_test)

print("Train feature names", Tf_Id_Vector.get_feature_names_out())
```

```
Train feature names ['aa' 'aa batteries' 'aaa' ... 'zipper' 'zits' 'zone']
```

5. Perceptron

In [24]:
```
perceptron = Perceptron(
    penalty= 'l1',       #Provided to model on wrong prediction
    alpha=0.000005,      #Strength of the L2 regularization term
    max_iter=1500,       #Maximum number of iterations over the data
    tol=1e-1,            #Tolerance for the optimization or the criterion for stopping
)

perceptron.fit(X_tf_id_train , y_train)

#prediction through X_test data
```

```
y_test_prediction=perceptron.predict(X_tf_id_test)

#Preparing the report by comparing actual value and predicted value

report=classification_report(y_test, y_test_prediction)

print("\n Values for Perceptron Model")
print(report)
```

```
 Values for Perceptron Model
              precision    recall  f1-score   support

           1       0.61      0.66      0.63      4029
           2       0.54      0.50      0.51      3990
           3       0.70      0.69      0.70      3981

    accuracy                           0.62     12000
   macro avg       0.62      0.62      0.61     12000
weighted avg       0.62      0.62      0.61     12000
```

6.SVM

In [26]:
```
svm = LinearSVC(
    C=0.10,                       #Regularization parameter. Default is 1.
    penalty='l2',                 #Norm of Penalty
    tol=1e-1,                     #tolerance of Stopping criteria. default is 1e-3
    class_weight="balanced",      #adjust and provides weight to each class
    max_iter=1000,                #Hard limit on iterations within solver, or -1 for no limit.
    random_state=1,               #Controls the pseudo random number generation for shuffling the data for probability estimates.
    loss='squared_hinge',         #Specifies the Loss Function
    dual=False,                   #Selects the algorithm to either the dual or primal optimization
    fit_intercept=False,          #Weather to calculate intercept for this model
)

svm.fit(X_tf_id_train, y_train)

#prediction through X_test data

y_test_prediction_svm=svm.predict(X_tf_id_test)

#Preparing the report by comparing actual value and predicted value

report_svm=classification_report(y_test, y_test_prediction_svm)

print("\n Values for SVM Model")

print(report_svm)
```

```
 Values for SVM Model
              precision    recall  f1-score   support

           1       0.69      0.71      0.70      4029
           2       0.60      0.56      0.58      3990
           3       0.74      0.78      0.76      3981

    accuracy                           0.68     12000
   macro avg       0.68      0.68      0.68     12000
weighted avg       0.68      0.68      0.68     12000
```

7. Logistic Regression

In [29]:
```
logistic_regression = LogisticRegression(
    penalty='l1',                 #Penalty to be given to model on wrong prediction
    tol=1e-2,                     #Tolerance of Stopping criteria
    C=0.7,                        #Inverse of Regularization
    random_state=1,               #Used to Shuffle data
    solver='saga',                #Algorithm to be used for optimzation step and to handle multinomial loss
    max_iter=2000,                #Max iterations to be taken for convergance
    multi_class='multinomial',    #loss minimised is the multinomial loss fit across the entire probability distribution
    warm_start=True,              #reuse the solution of the previous call to fit as initialization if set to true
)
logistic_regression.fit(X_tf_id_train , y_train)

#prediction through X_test data

y_test_prediction_log=logistic_regression.predict(X_tf_id_test)

#Preparing the report by comparing actual value and predicted value

report_logistic=classification_report(y_test, y_test_prediction_log)

print("\n Values for Logistic Regression Model")

print(report_logistic)
```

```
 Values for Logistic Regression Model
              precision    recall  f1-score   support

           1       0.69      0.71      0.70      4029
           2       0.60      0.58      0.59      3990
           3       0.76      0.76      0.76      3981

    accuracy                           0.68     12000
   macro avg       0.68      0.68      0.68     12000
weighted avg       0.68      0.68      0.68     12000
```

8. multinomial naive bayes

In [31]:
```
naive_bayes = MultinomialNB(
        alpha=2,                          #Additive (Laplace/Lidstone) smoothing parameter
        fit_prior=False,                  #to learn class prior probabilities or not
        class_prior=[1.4, 1.6, 1.8]       #Prior probilities of class, not adjusted according to data
)

naive_bayes.fit(X_tf_id_train , y_train)

#prediction through X_test data

y_test_prediction_nb=naive_bayes.predict(X_tf_id_test)

#Preparing the report by comparing actual value and predicted value

report_nb=classification_report(y_test, y_test_prediction_nb)

print("\n Values for Multinomial Naive Bayes")
print(report_nb)
```

```
Values for Multinomial Naive Bayes
              precision    recall  f1-score   support

           1       0.74      0.61      0.67      4029
           2       0.58      0.61      0.59      3990
           3       0.71      0.80      0.75      3981

    accuracy                           0.67     12000
   macro avg       0.68      0.67      0.67     12000
weighted avg       0.68      0.67      0.67     12000
```

In [ ]: