In [1]:
```python
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('punkt')
import re
from bs4 import BeautifulSoup
import contractions
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import confusion_matrix as cm
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report
from sklearn.metrics import precision_recall_fscore_support
import warnings
warnings.filterwarnings("ignore")
```

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

In [2]:
```python
# ! 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]:
```python
df = pd.read_csv('./amazon_reviews_us_Beauty_v1_00.tsv', sep='\t', error_bad_lines=False, warn_bad_lines=False)
```

Keep Reviews and Ratings

In [4]:
```python
df = df[['star_rating', 'review_body']]
```

let ratings with the values of 1 and 2 form class 1, ratings with the value of 3 form class 2, and ratings with the values of 4 and 5 form class 3

In [5]:
```python
class_one = df[(df['star_rating']==1) | (df['star_rating']==2)]
class_two = df[df['star_rating']==3]
class_three = df[(df['star_rating']==4) | (df['star_rating']==5)]

class_one.loc[:, "label"] =1
class_two.loc[:, "label"] =2
class_three.loc[:, "label"] =3
```

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

In [30]:
```python
class_one = class_one.sample(n=20000, random_state=100)
class_two = class_two.sample(n=20000, random_state=100)
class_three = class_three.sample(n=20000, random_state=100)
dataset = pd.concat([class_one, class_two, class_three])
dataset.reset_index(drop=True)
train = dataset.sample(frac=0.8, random_state=100)
test = dataset.drop(train.index)
train = train.reset_index(drop = True)
test = test.reset_index(drop = True)
```

Data Cleaning

Ans: For DATA CLEANING:

- First we calculate average length of a review before the cleaning
- We then convert each review string to lowercase by using str.lower()
- From the review body, to remove HTML elements we used BeautifulSoup and for the removal of the URLs in the review we used regular expression which eliminates the URLs from a review, for the process to remove the HTML elements and URLs, we clean each review by using lambda function and the processed review is then reassigned to the particular position in the dataframe by using the '.apply' function
- The BeautifulSoup(review, 'html.parser') returns an object which has parsed html and remaining text and we then extract the remaining text
- Non alphabetical characters and extra spaces are substituted by single space ' ' by the help of re.sub() , '[^a-zA-Z]+' means the characters not involving alphabets.
- Contractions are removed by the use of Contractions library which takes each review from the dataframe and expands contraction with the help of contractions.fix() function
- At the end, calculate average length of a review after the cleaning

convert all reviews into lowercase

In [7]:
```
avg_len_before = (train['review_body'].str.len().sum() + test['review_body'].str.len().sum())/60000

#Covert all reviews to lower case
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 [8]:
```
# URL Remover code
train['review_body'] = train['review_body'].apply(lambda x: re.split('https:\/\/.*', str(x))[0])
test['review_body'] = test['review_body'].apply(lambda x: re.split('https:\/\/.*', str(x))[0])

def html_tag_remover(review):
    soup = BeautifulSoup(review, 'html.parser')
    review = soup.get_text()
    return review

train['review_body'] = train['review_body'].apply(lambda review: html_tag_remover(review))
test['review_body'] = test['review_body'].apply(lambda review: html_tag_remover(review))
```

remove non-alphabetical characters

In [9]:
```
train['review_body'] = train['review_body'].apply(lambda review: re.sub('[^a-zA-Z]+',' ', review))
test['review_body'] = test['review_body'].apply(lambda review: re.sub('[^a-zA-Z]+',' ', review))
```

remove extra spaces

In [10]:
```
train['review_body'] = train['review_body'].apply(lambda review: re.sub(' +', ' ', review))
test['review_body'] = test['review_body'].apply(lambda review: re.sub(' +', ' ', review))
```

perform contractions on the reviews

In [28]:
```
def expand_contractions(review):
    review = contractions.fix(review)
    return review

train['review_body'] = train['review_body'].apply(lambda review: expand_contractions(review))
test['review_body'] = test['review_body'].apply(lambda review: expand_contractions(review))
avg_len_after = (train['review_body'].str.len().sum() + test['review_body'].str.len().sum())/60000

print('Average length of the reviews in terms of character length before and after cleaning: ',
      avg_len_before, ",",avg_len_after)
```

```
Average length of the reviews in terms of character length before and after cleaning:  287.3779333333333 , 165.6902
```

Pre-processing

Ans: For DATA PREPROCESSING:

- First we calculate average length of a review before the preprocessing
- I used NLTk library for stopward removal and to perform lemmatization
- In the stop removal snippet, for each review of the dataframe, we first get all the stopwords in englush and then we generate tokens of a review with the word_tokenize() function and then we create the list of words which doesnot contain the stopwords and then apply the filtered review back to the dataframe in the form of list.

- In the lemmatization step, for each review of the dataframe, we first lemmatize each and every word of the review and then create the string of the final lemmatized words of the review and apply back to the dataframe
- At the end, calculate average length of a review after the preprocessing

remove the stop words

```
In [12]: avg_len_before_prepro = (train['review_body'].str.len().sum() + test['review_body'].str.len().sum())/60000
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize

def remove_stopwords(review):
    stop_words_english = set(stopwords.words('english'))
    review_word_tokens = word_tokenize(review)
    filtered_review = [word for word in review_word_tokens if not word in stop_words_english]
    return filtered_review

train['review_body'] = train['review_body'].apply(lambda review: remove_stopwords(review))
test['review_body'] = test['review_body'].apply(lambda review: remove_stopwords(review))
```

perform lemmatization

```
In [29]: from nltk.stem import WordNetLemmatizer

def review_lemmatize(review):
    lemmatizer = WordNetLemmatizer()
    lemmatized_review = [lemmatizer.lemmatize(word) for word in review]
    return ' '.join(lemmatized_review)

train['review_body'] = train['review_body'].apply(lambda review: review_lemmatize(review))
test['review_body'] = test['review_body'].apply(lambda review: review_lemmatize(review))

avg_len_after_prepro = (train['review_body'].str.len().sum() + test['review_body'].str.len().sum())/60000

print('Average length of the reviews in terms of character length before and after preprocessing: ',
      avg_len_before_prepro, ",", avg_len_after_prepro)
```

```
Average length of the reviews in terms of character length before and after preprocessing:  276.02595 , 330.3811666666667
```

TF-IDF Feature Extraction

Ans: For TF-IDF Feature Extraction

- we used TfidfVectorizer library to convert each review into feature vectors
- we used min_df to When building the vocabulary ignore terms that have a document frequency strictly lower than the given 0.01
- Here by using fit_transform() it learns vocabulary and idf and return document-term matrix which we then convert to get a new dataframe of vectors. We do the same procedure to get the vectors for training and testing data

```
In [14]: train_corpus = list(train['review_body'])
test_corpus = list(test['review_body'])
tfidf_vectorizer = TfidfVectorizer(min_df = 0.001)
X_train = tfidf_vectorizer.fit_transform(train_corpus)
X_train = pd.DataFrame(X_train.toarray())
X_test = tfidf_vectorizer.transform(test_corpus)
X_test = pd.DataFrame(X_test.toarray())
Y_train = train['label']
Y_test = test['label']
Y_train = Y_train.astype('int')
Y_test = Y_test.astype('int')
```

Perceptron

- Perceptron model was used and imported from sklearn library, we set random_state to 100 and eta0 to 0.1. We trained the training data using .fit() method and predicted the output for the test data

```
In [24]: perceptr = Perceptron(random_state = 100, eta0=0.1)
perceptr.fit(X_train, Y_train)
Y_pred_test = perceptr.predict(X_test)

percep_scores = precision_recall_fscore_support(Y_test, Y_pred_test, average=None)
average_percep_scores = precision_recall_fscore_support(Y_test, Y_pred_test, average='macro')

print("\nPrecision, Recall, and f1-score for the testing split in 4 lines for Perceptron")
print('Class 1: %f, %f, %f' % (percep_scores[0][0], percep_scores[1][0], percep_scores[2][0]))
print('Class 2: %f, %f, %f' % (percep_scores[0][1],  percep_scores[1][1],  percep_scores[2][1]))
print('Class 3: %f, %f, %f' % (percep_scores[0][2],  percep_scores[1][2],  percep_scores[2][2]))
print('Average: %f, %f, %f' % (average_percep_scores[0], average_percep_scores[1], average_percep_scores[2]))
```

```
Precision, Recall, and f1-score for the testing split in 4 lines for Perceptron
Class 1: 0.696101, 0.594079, 0.641056
Class 2: 0.565405, 0.461210, 0.508020
Class 3: 0.596618, 0.798982, 0.683128
Average: 0.619375, 0.618090, 0.610735
```

SVM

- SVM model was used and imported from sklearn library, we set random_state to 100, max_iter to 1000. We trained the training data using .fit() method and predicted the output for the test data

In [25]:
```
clf_svm = LinearSVC(random_state=100, max_iter=1000)
clf_svm.fit(X_train, Y_train)
Y_pred_test_svm = clf_svm.predict(X_test)

svm_scores = precision_recall_fscore_support(Y_test, Y_pred_test_svm, average=None)
average_svm_scores = precision_recall_fscore_support(Y_test, Y_pred_test_svm, average='macro')

print("\nPrecision, Recall, and f1-score for the testing split in 4 lines for SVM")
print('Class 1: %f, %f, %f' % (svm_scores[0][0], svm_scores[1][0], svm_scores[2][0]))
print('Class 2: %f, %f, %f' % (svm_scores[0][1],  svm_scores[1][1],  svm_scores[2][1]))
print('Class 3: %f, %f, %f' % (svm_scores[0][2],  svm_scores[1][2],  svm_scores[2][2]))
print('Average: %f, %f, %f' % (average_svm_scores[0], average_svm_scores[1], average_svm_scores[2]))
```

```
Precision, Recall, and f1-score for the testing split in 4 lines for SVM
Class 1: 0.689001, 0.701982, 0.695431
Class 2: 0.602248, 0.551594, 0.575809
Class 3: 0.705826, 0.752163, 0.728258
Average: 0.665692, 0.668580, 0.666499
```

Logistic Regression

- Logistic Regression model was used and imported from sklearn library, we set random_state to 100, max_iter to 1000. We trained the training data using .fit() method and predicted the output for the test data

In [26]:
```
logistic = LogisticRegression(random_state = 100, max_iter=1000)
logistic.fit(X_train, Y_train)
Y_pred_test_logis = logistic.predict(X_test)

logistic_scores = precision_recall_fscore_support(Y_test, Y_pred_test_logis, average=None)
average_logistic_scores = precision_recall_fscore_support(Y_test, Y_pred_test_logis, average='macro')

print("\nPrecision, Recall, and f1-score for the testing split in 4 lines for Logistic Regression")
print('Class 1: %f, %f, %f' % (logistic_scores[0][0], logistic_scores[1][0], logistic_scores[2][0]))
print('Class 2: %f, %f, %f' % (logistic_scores[0][1],  logistic_scores[1][1],  logistic_scores[2][1]))
print('Class 3: %f, %f, %f' % (logistic_scores[0][2],  logistic_scores[1][2],  logistic_scores[2][2]))
print('Average: %f, %f, %f' % (average_logistic_scores[0], average_logistic_scores[1], average_logistic_scores[2]))
```

```
Precision, Recall, and f1-score for the testing split in 4 lines for Logistic Regression
Class 1: 0.694044, 0.707120, 0.700521
Class 2: 0.600364, 0.580467, 0.590248
Class 3: 0.730489, 0.740712, 0.735565
Average: 0.674966, 0.676100, 0.675445
```

Naive Bayes

- Naive Bayes model was used and imported from sklearn library, we set random_state to 100, max_iter to 1000. We trained the training data using .fit() method and predicted the output for the test data

In [27]:
```
naive_bay = MultinomialNB(force_alpha=True)
naive_bay.fit(X_train, Y_train)
Y_pred_test_naive = logistic.predict(X_test)

naive_scores = precision_recall_fscore_support(Y_test, Y_pred_test_naive, average=None)
average_naive_scores = precision_recall_fscore_support(Y_test, Y_pred_test_naive, average='macro')


print("\nPrecision, Recall, and f1-score for the testing split in 4 lines for Naive Bayes")
print('Class 1: %f, %f, %f' % (naive_scores[0][0], naive_scores[1][0], naive_scores[2][0]))
print('Class 2: %f, %f, %f' % (naive_scores[0][1],  naive_scores[1][1],  naive_scores[2][1]))
print('Class 3: %f, %f, %f' % (naive_scores[0][2],  naive_scores[1][2],  naive_scores[2][2]))
print('Average: %f, %f, %f' % (average_naive_scores[0], average_naive_scores[1], average_naive_scores[2]))
```

```
Precision, Recall, and f1-score for the testing split in 4 lines for Naive Bayes
Class 1: 0.694044, 0.707120, 0.700521
Class 2: 0.600364, 0.580467, 0.590248
Class 3: 0.730489, 0.740712, 0.735565
Average: 0.674966, 0.676100, 0.675445
```