Import libraries

Extra library added are contractions and Sklearn

In [1]:
```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
from bs4 import BeautifulSoup
import contractions
import nltk
nltk.download('stopwords')
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import classification_report, f1_score, recall_score, precision_sc
from sklearn.linear_model import Perceptron
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
import warnings
warnings.filterwarnings('ignore')
```

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

```
ERROR: Invalid requirement: '#'
```

Read Data

In [3]:
```
data = pd.read_csv('data.tsv',sep='\t',on_bad_lines='skip', header=0, quotechar='"', dt
```

The data given is read with the help of pandas read_csv using the compression gzip as data is compressed. Further I have used '\t' as seperator to format data.

Keep Reviews and Ratings

As mentioned below cell gets only 'review_body' and 'star_rating' columns from the whole data. Both the columns are merged and stored further

In [4]:
```
getData = data[['review_body','star_rating']]
getData
```

Out[4]:

	review_body	star_rating
0	Love this, excellent sun block!!	5
1	The great thing about this cream is that it do...	5
2	Great Product, I'm 65 years old and this is al...	5
3	I use them as shower caps & conditioning caps....	5
4	This is my go-to daily sunblock. It leaves no ...	5
...	...	...
5094302	After watching my Dad struggle with his scisso...	5
5094303	Like most sound machines, the sounds choices a...	3
5094304	I bought this product because it indicated 30 ...	5
5094305	We have used Oral-B products for 15 years; thi...	5
5094306	I love this toothbrush. It's easy to use, and ...	5

5094307 rows × 2 columns

Next the cell check for the NaN values for both the columns

Review body has 400 NaN values and Star rating has 10 NaN values

In [5]:
```
getData.isnull().sum()
```

Out[5]:
```
review_body    400
star_rating     10
dtype: int64
```

As the NaN values are less compared to the total dataset rows, I therefore dropped the NaN values from both the columns

In [6]:
```
getData=getData.dropna()
```

After dropping the NaN values below cell confirms that there are no further NaN values in the data

In [7]:
```
getData.isnull().sum()
```

Out[7]:
```
review_body    0
star_rating    0
dtype: int64
```

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

label_class funtion helps to to add designated class to star_rating. In this a extra column is created with name class which stores the value of class with respect to star_rating

Star_rating: 1, 2 Class: 1

Star_rating: 3 Class: 2

Star_rating: 4, 5 Class: 3

In [8]:
```
pd.options.mode.chained_assignment = None
def labelClass(rating):
    if rating == "1" :
          return 1
    if rating == "2" :
          return 1
    if rating == "3" :
          return 2
    if rating == "4":
          return 3
    if rating  == "5":
          return 3
getData['class'] = getData['star_rating'].map(labelClass)
```

Ramdom 20000 data rows from all three columns(review_body, star_rating, class) gets selected

In [9]:
```
classOne = getData.loc[getData['class'] == 1].sample(n=20000)
classTwo = getData.loc[getData['class'] == 2].sample(n=20000)
classThree = getData.loc[getData['class'] == 3].sample(n=20000)
```

The 20000 randomly slected data from all the three classes are further merged into one data providing the total rows of 60000

In [10]:
```
getRandomData = pd.concat([classOne, classTwo, classThree])
getRandomData
```

Out[10]:

	review_body	star_rating	class
4288031	there are two major problems with this bag. 1 ...	1	1
1146349	smell gave me a migraine	1	1
342233	The color was more orange than red. And, it o...	1	1
2780898	It is artificial scent & don't feel butter. I ...	2	1
4690197	Let's face it, \\"aluminum\\" deodorant produc...	2	1
...	...	...	...
852671	I am overly happy with this new nail dryer. I...	5	3
3012545	Worked to relieve my scalp itch - my dermatolo...	4	3
1985815	works great! best and cheapest brush set ever	5	3
4183444	I tried several weeks of whitening from a kit ...	5	3
3376191	My wife bought this for me for Christmas to fu...	4	3

60000 rows × 3 columns

In [11]:
```
getRandomData['review_body'] = getRandomData['review_body'].astype(str)
```

Data Cleaning

average length of review's are recorded before data cleaning

In [12]:
```
avgLenBeforeClean = getRandomData['review_body'].apply(len).mean()
```

various data cleaning strategies are are applied to clean and process the review data

1) all the reviews are converted to lower case

2) HTML and URL's are removed from the reviews

3) Last contractions are performed on the review's this will change won't → will not, I'm → i am and so on with the use of contraction library

4) Extra spaces are removed

5) Non alphabetical characters are also removed

In [13]:
```
getRandomData['review_body'] = getRandomData['review_body'].apply(str.lower)
getRandomData['review_body'] = getRandomData['review_body'].apply(lambda x: re.sub(re.c
getRandomData['review_body'] = getRandomData['review_body'].apply(lambda x: re.sub(re.c
getRandomData['review_body'] = getRandomData['review_body'].apply(lambda x: re.sub(' +'
```

In [14]:
```
def contra(text):
    words = []
    for word in text.split():
        words.append(contractions.fix(word))
    word_text = ' '.join(words)
    return word_text
getRandomData['review_body'] = getRandomData['review_body'].apply(contra)
```

In [15]:
```
getRandomData['review_body'] = getRandomData['review_body'].apply(lambda x: re.sub(re.c
```

In [16]:
```
avgLenAfterClean = getRandomData['review_body'].apply(len).mean()
```

In [17]:
```
print('Average length of review_body before and after data cleaning is '+ str(avgLenBef
```

```
Average length of review_body before and after data cleaning is 269.57765, 267.771166666
66666
```

Pre-processing

remove the stop words

average length of review's are recorded before data pre-processing

In [18]:
```
avgLenBeforePre = getRandomData['review_body'].apply(len).mean()
```

Stop words are the commonly used words which does not create much impact in a sentance and ignoring such words will ease in creating the model for prediction

below cell removes the stop words from the review_body column by using the nltk.corpus library

In [19]:
```
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
getRandomData['review_body'] = getRandomData['review_body'].apply(lambda x: ' '.join([w
```

perform lemmatization

lemmatization can be said as the step done to group toegther same type of words, to remove inflectional words and to return the base word

below cell does the lemmatization by using the nltk.stem library and importing WordNetLemmatizer

In [20]:
```
from nltk.stem import WordNetLemmatizer
w_tokenizer = nltk.tokenize.WhitespaceTokenizer()
lemmatizer = nltk.stem.WordNetLemmatizer()

def lemmatize_text(text):
    return " ".join([lemmatizer.lemmatize(w) for w in w_tokenizer.tokenize(text)])

getRandomData['review_body'] = getRandomData['review_body'].apply(lemmatize_text)
```

In [21]:
```
avgLenAfterPre = getRandomData['review_body'].apply(len).mean()
```

In [22]:
```
print('Average length of review_body before and after pre-processing is '+ str(avgLenBe
```

```
Average length of review_body before and after pre-processing is 267.77116666666666, 15
5.93403333333333
```

TF-IDF Feature Extraction

Term Frequency – Inverse Document Frequency, this shows that how much important a word or phrase to a given document. We can use this technique

when we have a Bag of words and we need to extract some information.

In [23]:
```
X = getRandomData['review_body']
y = getRandomData['class']

vectorizer = TfidfVectorizer()
X = vectorizer.fit_transform(X)
```

After TF-IDF Feature Extraction final data is splitted with 20% test size and 80% train size with the help of train_test_split using random_state = 10

In [24]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state
```

Perceptron

Grid search cross validation is used to further train the perceptron model on various parameter of max_iter, alpha, penalty and eta0

This will help to get the exact value of parameters which can be used with the final perceptron model. This will help to achieve higher accuracy.

In [25]:
```
params = {'max_iter': [50, 100, 500, 1000], 'penalty': ['l2', 'l1', 'elasticnet'], 'eta
gridPermodel = GridSearchCV(Perceptron(), params, cv=5).fit(X_train, y_train)
perceptronModel = Perceptron(max_iter = gridPermodel.best_params_['max_iter'], eta0 = g
y_predict = perceptronModel.predict(X_test)
```

Generating classification report for perceptron model and printing precision, recall and f1-score for all the three classes.

In [26]:
```
perceptronCR = classification_report(y_predict,y_test, output_dict=True)
print(classification_report(y_predict,y_test))
```

```
              precision    recall  f1-score   support

           1       0.54      0.64      0.58      3316
           2       0.61      0.49      0.54      4976
           3       0.64      0.69      0.66      3708

    accuracy                           0.60     12000
   macro avg       0.60      0.61      0.60     12000
weighted avg       0.60      0.60      0.59     12000
```

In [27]:
```
print('classification report of Perceptron')
print('Class        Precision            Recall             F1-score')
print('1'+"      "+ str(perceptronCR['1']['precision'])+",   "+str(perceptronCR['1']['r
print('2'+"      "+ str(perceptronCR['2']['precision'])+",   "+str(perceptronCR['2']['r
print('3'+"      "+ str(perceptronCR['3']['precision'])+",   "+str(perceptronCR['3']['r
print('average' +" "+str((perceptronCR['1']['precision']+perceptronCR['2']['precision']
```

```
classification report of Perceptron
Class        Precision            Recall             F1-score
```

```
1       0.5366591080876795,   0.6423401688781665,   0.5847632120796157
2       0.6123933768188661,   0.49055466237942125,   0.5447444766793127
3       0.6363411619283066,   0.6941747572815534,   0.6640010318586355
average 0.5951312156116174, 0.6090231961797138, 0.5978362402058547
```

SVM

Grid search cross validation is used to further train the SVM model on parameter of C

This will help to get the exact value of C which can be used with the final SVM model. This will help to achieve higher accuracy.

In [28]:
```
params = {"C": [0.01,0.1,1,10,100,1000,10000]}
gridSVMmodel = GridSearchCV(LinearSVC(), params, cv=5).fit(X_train, y_train)
svmModel = LinearSVC(C = gridSVMmodel.best_params_['C']).fit(X_train, y_train)
svmModel = LinearSVC().fit(X_train, y_train)
y_predict = svmModel.predict(X_test)
```

Generating classification report for SVM model and printing precision, recall and f1-score for all the three classes.

In [29]:
```
svmCR = classification_report(y_predict,y_test, output_dict=True)
print(classification_report(y_predict,y_test))
```

```
              precision    recall  f1-score   support

           1       0.68      0.66      0.67      4134
           2       0.54      0.58      0.56      3714
           3       0.75      0.73      0.74      4152

    accuracy                           0.66     12000
   macro avg       0.66      0.66      0.66     12000
weighted avg       0.66      0.66      0.66     12000
```

In [30]:
```
print('classification report of SVM')
print('Class        Precision            Recall            F1-score')
print('1'+"      "+ str(svmCR['1']['precision'])+",   "+str(svmCR['1']['recall'])+",
print('2'+"      "+ str(svmCR['2']['precision'])+",   "+str(svmCR['2']['recall'])+",
print('3'+"      "+ str(svmCR['3']['precision'])+",   "+str(svmCR['3']['recall'])+",
print('average' +" "+str((svmCR['1']['precision']+svmCR['2']['precision']+svmCR['3']['p
```

```
classification report of SVM
Class        Precision            Recall            F1-score
1       0.6835474930713026,   0.6562651185292695,   0.6696285326422312
2       0.5423983943803311,   0.5821217016693592,   0.5615584415584415
3       0.74610630407911,   0.726878612716763,   0.7363669635232403
average 0.657350730510248, 0.6550884776384639, 0.6558513125746376
```

Logistic Regression

Grid search cross validation is used to further train the logistic regression model on parameter of penalty

This will help to get the exact value of parameter penalty which can be used with the final logistic regression model. This will help to achieve higher accuracy.

In [31]:
```
params = {"penalty": ['l1','l2','elasticnet',None]}
gridLRmodel = GridSearchCV(LogisticRegression(), params, cv=5).fit(X_train, y_train)
logRegModel = LogisticRegression(penalty = gridLRmodel.best_params_['penalty'], solver=
y_predict = logRegModel.predict(X_test)
```

Generating classification report for logistic regression model and printing precision, recall and f1-score for all the three classes.

In [32]:
```
logRegCR = classification_report(y_predict,y_test, output_dict=True)
print(classification_report(y_predict,y_test))
```

```
              precision    recall  f1-score   support

           1       0.70      0.67      0.68      4123
           2       0.58      0.59      0.58      3890
           3       0.74      0.76      0.75      3987

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

In [33]:
```
print('classification report of Logistic Regression')
print('Class        Precision            Recall             F1-score')
print('1'+"      "+ str(logRegCR['1']['precision'])+",   "+str(logRegCR['1']['recall'])
print('2'+"      "+ str(logRegCR['2']['precision'])+",   "+str(logRegCR['2']['recall'])
print('3'+"      "+ str(logRegCR['3']['precision'])+",   "+str(logRegCR['3']['recall'])
print('average' +" "+str((logRegCR['1']['precision']+logRegCR['2']['precision']+logRegC
```

```
classification report of Logistic Regression
Class        Precision            Recall             F1-score
1      0.6981607457797934,   0.6720834343924327,   0.6848739495798319
2      0.5772704465629704,   0.5915167095115681,   0.5843067546978161
3      0.7446229913473424,   0.7554552294958615,   0.7500000000000001
average 0.6733513945633688, 0.6730184577999542, 0.6730602347592161
```

Naive Bayes

Grid search cross validation is used to further train the naive bayes model on various parameter of alpha

This will help to get the exact value of parameter alpha which can be used with the final naive bayes model. This will help to achieve higher accuracy.

In [34]:
```
params = {'alpha': [0.01,0.1,0,1.0,10.0]}
gridNBmodel = GridSearchCV(MultinomialNB(), params, cv=5).fit(X_train, y_train)
naiBayModel = MultinomialNB(alpha = gridNBmodel.best_params_['alpha']).fit(X_train, y_t
y_predict = naiBayModel.predict(X_test)
```

Generating classification report for naive bayes model and printing precision, recall and f1-score for all the three classes.

In [35]:
```
naiBayCR = classification_report(y_predict,y_test, output_dict=True)
print(classification_report(y_predict,y_test))
```

```
              precision    recall  f1-score   support

           1       0.64      0.68      0.66      3752
           2       0.63      0.55      0.59      4573
           3       0.69      0.76      0.72      3675

    accuracy                           0.65     12000
   macro avg       0.65      0.66      0.66     12000
weighted avg       0.65      0.65      0.65     12000
```

In [36]:
```
print('classification report of Naive Bayes')
print('Class        Precision            Recall              F1-score')
print('1'+"      "+ str(naiBayCR['1']['precision'])+",    "+str(naiBayCR['1']['recall'])
print('2'+"      "+ str(naiBayCR['2']['precision'])+",    "+str(naiBayCR['2']['recall'])
print('3'+"      "+ str(naiBayCR['3']['precision'])+",    "+str(naiBayCR['3']['recall'])
print('average' +" "+str((naiBayCR['1']['precision']+naiBayCR['2']['precision']+naiBayC
```

```
classification report of Naive Bayes
Class        Precision            Recall              F1-score
1      0.6417233560090703,   0.6788379530916845,   0.6597590985623624
2      0.631961866532865,   0.5508418980975289,   0.5886201659072321
3      0.6860321384425216,   0.7551020408163265,   0.7189119170984456
average 0.6532391203281523, 0.6615939640018466, 0.6557637271893467
```

In [ ]:
```
```