CSCI544 - Homework Assignment No 1

1. Dataset Preparation

1.Imported the data as a Pandas frame using Pandas package and only kept the Reviews and Ratings fields in the input data frame to generate data

```
Dataset=pd.read_table('../amazon_reviews_us_Beauty_v1_00.tsv.gz',on_bad_lines='skip')
InputDataFrame = dataset[['star_rating','review_body']].copy()
```

2.Dropped the column with the null value and converted the datatype of star_rating column to int for consistency

```
InputDataFrame.dropna(inplace = True)
InputDataFrame['star_rating']=InputDataFrame['star_rating'].astype('float').astype('int')
```

3.Create a three-class classification problem according to ratings with
ratings with the values of 1 and 2 from class 1,
ratings with the value of 3 form class 2, and
ratings with the values of 4 and 5 form class 3.

```
def assign_class(value):
    if value == 1 or value == 2:
        return 1
    if value == 3:
        return 2
    if value == 4 or value == 5:
        return 3

InputDataFrame['class'] = InputDataFrame['star_rating'].map(assign_class)
```

4.To avoid the computational burden, select 20,000 random reviews from each rating class and create a balanced dataset to perform the required tasks on the downsized dataset.

```
InputDataFrame_1 = InputDataFrame.loc[InputDataFrame['class'] == 1].sample(n = 20000)
InputDataFrame_2 = InputDataFrame.loc[InputDataFrame['class'] == 2].sample(n = 20000)
InputDataFrame_3 = InputDataFrame.loc[InputDataFrame['class'] == 3].sample(n = 20000)
InputDataFrames = [InputDataFrame_1, InputDataFrame_2, InputDataFrame_3]
InputDataFrameFinal = pd.concat(InputDataFrames)
```

2. Data Cleaning

1.Data cleaning steps to preprocess the dataset you created

Average length of the reviews in terms of character length of dataset before clean - 268.7442

- code to convert all reviews into lowercase where each word of the data frame is lowered with the help of python function .lower()

```
class Lowercase(DataCleaning):
  def clean(self) -> None:
    InputDataFrameFinal['review_body'] =
InputDataFrameFinal['review_body'].apply(lambda text: str(text).lower())
```

- code to remove the HTML and URLs from the reviews where two different regex is used to remove the html tag and the url by passing each review text to html_url function

```
class RemoveHTMLURL(DataCleaning):
  def html_url(self, review):
    review = re.sub('https?://\S+|www\.\S+', '', review) # html
    review = re.sub('<[^<]+?>', '', review)               # url
    return review

  def clean(self) -> None:
    InputDataFrameFinal['review_body'] =
InputDataFrameFinal['review_body'].apply(lambda text: self.html_url(text))
```

- code to remove non-alphabetical characters where single regex expression which remove the characters except the a-z, A-Z and spaces between words

```
 class RemoveNonAlphabeticalCharacter(DataCleaning):
   def clean(self) -> None:
     InputDataFrameFinal['review_body'] =
 InputDataFrameFinal['review_body'].apply(lambda text:re.sub('[^a-zA-Z\s]','', text))
```

- code to remove extra spaces where regex is used to remove extra white spaces

```
class RemoveExtraSpaces(DataCleaning):
  def clean(self) -> None:
    InputDataFrameFinal['review_body'] =
InputDataFrameFinal['review_body'].apply(lambda text: re.sub(' +', ' ', text))
```

- code to perform contractions on the reviews where different regexes are used to decontract the words in the review. Two general contraction regex are used and 8 specific regexes for contraction are used.

```
class Contraction(DataCleaning):
  def contraction_function(self, review):
      review = re.sub(r"won\'t", "will not", review)
      review = re.sub(r"can\'t", "can not", review)

      review = re.sub(r"n\'t", " not", review)
      review = re.sub(r"\'re", " are", review)
      review = re.sub(r"\'s", " is", review)
      review = re.sub(r"\'d", " would", review)
      review = re.sub(r"\'ll", " will", review)
      review = re.sub(r"\'t", " not", review)
      review = re.sub(r"\'ve", " have", review)
      review = re.sub(r"\'m", " am", review)
      return review

  def clean(self) -> None:
      InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body']
                             .apply(lambda text:self.contraction_function(text))
```

Average length of the reviews in terms of character length of dataset after clean - 257.97265

3. Preprocessing

Using NLTK package to process the dataset by remove the stop words and tokenizing the reviews and applying part of speech tagging technique to identify part of speech for each word in the review for accurate performance of lemmatization

Average length of the reviews in terms of character length of dataset before preprocessing - 257.97265

- code to remove the stop words by comparing each word in the review text to the list of stop words from the nltk english stopword library and discarding the word from the review if present

```
nltk.download('stopwords')
stop_words=stopwords.words('english')
```

```
def remove_stopword() -> None:
      InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body']
                       .apply(lambda text: ' '.join([word for word in text.split()
                             if word not in (stop_words)]))
```

- removal of stopwords resulted in a lower performance score. Hence for the final preprocessing, stopword removal was not performed and only lemmatization was performed.

- code to perform lemmatization where each review text is split into list of words and each word is assigned a part of speech tag and based on the part of speech tag each word is lemmatized with the help of WordNetLemmatizer of nltk

```
def lemmatization(review) -> None:
  lemmatizer = WordNetLemmatizer()
  lemmatized_sentence = []
  for word, tag in pos_tag(review):
    if (review is None):
        return review
    else:
        if tag.startswith('NN'):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'
        lemmatized_sentence.append(lemmatizer.lemmatize(word, pos))
  return lemmatized_sentence

InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body']
                             .apply(lambda text: ' '.join(lemmatization(text.split())))
```

Average length of the reviews in terms of character length of dataset after preprocessing - 248.368967

4. Feature Extraction

Using sklearn to extract TF-IDF features with ngram range of (1,4)and storing the feature vector in X and class output value in Y

```
TF_IDF = TfidfVectorizer(ngram_range=(1,4))
```

```
X = TF_IDF.fit_transform(InputDataFrameFinal["review_body"])
Y = InputDataFrameFinal['class']
```

Finally the dataset(X,Y) is created using train_test_split function of sklearn that consists of features X and labels Y for the reviews selected.

```
x_train,x_test,y_train,y_test=train_test_split(X, Y, test_size=0.2, random_state=100)
X_train.shape - (48000, 1991907)
X_test.shape - (12000, 1991907)
Y_train.shape - (48000,)
Y_test.shape - (12000,)
```

4. Perceptron

```
model_perceptron = Perceptron()
model_perceptron.fit(x_train,y_train)
```

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

```
                Precision              Recall            F1-score
Class 1 - 0.7233779098563645, 0.7352126856279889, 0.7292472849831482
Class 2 - 0.6385936222403925, 0.6024685008999743, 0.6200052924053983
Class 3 - 0.7943163289075239, 0.8240695988400193, 0.8089194638832878
Average - 0.7187626203347603, 0.7205835951226609, 0.7193906804239448
```

5. SVM

```
model_svm = LinearSVC()
model_svm.fit(x_train,y_train))
```

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

```
                Precision              Recall            F1-score
Class 1 - 0.7561412487205732, 0.7437704505411528, 0.7499048344118767
Class 2 - 0.6537467700258398, 0.6505528413473901, 0.6521458950895734
Class 3 - 0.8164377072477499, 0.8330111164813919, 0.8246411483253588
Average - 0.7421085753313875, 0.7424448027899783, 0.7422306259422697
```

6. Logistic Regression

```
model_logistic_regression=LogisticRegression(max_iter = 10000)
model_logistic_regression.fit(x_train,y_train)
```

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

	Precision	Recall	F1-score
Class 1 -	0.7492323439099283,	0.7369745784042285,	0.7430529120669966
Class 1 -	0.644544997486174,	0.6592954487014657,	0.65183678657684
Class 1 -	0.8128342245989305,	0.8081198646689222,	0.81047018904508
Average -	0.7355371886650109,	0.7347966305915388,	0.7351199625629722

7. Multinomial Naive Bayes

```
model_naive_bayes = MultinomialNB()
model_naive_bayes.fit(x_train,y_train)
```

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

	Precision	Recall	F1-score
Class 1 -	0.7613969608104505,	0.7188522527057639,	0.739513205592957
Class 2 -	0.5889044102973459,	0.7588068912316791,	0.6631460674157303
Class 3 -	0.900555898702903,	0.7046882551957467,	0.7906724511930586
Average -	0.7502857566035664,	0.7274491330443965,	0.7311105747339153

In [ ]:

```
import pandas as pd
import numpy as np
import nltk
nltk.download('wordnet')
import re
import string
from bs4 import BeautifulSoup


from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron
from sklearn.naive_bayes import MultinomialNB
from nltk.metrics.scores import precision
import sklearn.metrics as metrics
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
```

In [ ]:

```
# ! 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
```

Helper Function

In [ ]:

```
# Average Length Function
def average_length(review_column) -> str:
  return review_column.apply(len).mean()

def average_length_print(before_cleaning, review_column) -> str:
  return str(before_cleaning) + ", " + str(review_column.apply(len).mean())
```

Read Data

In [6]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

In [ ]:

```
## Reference - https://www.geeksforgeeks.org/how-to-load-a-tsv-file-into-a-pandas-dataframe/

# Read the data as a Pandas frame using Pandas package and only keep the Reviews and Ratings fields in the input data frame to generate data
dataset = pd.read_table('/content/drive/MyDrive/Applied Natural Language Processing/Homework1/amazon_reviews_us_Beauty_v1_00.tsv.gz', on_bad_lines='skip')
```

Keep Reviews and Ratings

In [ ]:

```
## Reference - https://www.statology.org/pandas-create-dataframe-from-existing-dataframe/

InputDataFrame = dataset[['star_rating','review_body']].copy()
InputDataFrame.to_csv('/content/drive/MyDrive/Applied Natural Language Processing/Homework1/Dataset_Input_Data_Frame.csv', index=False)
# display(InputDataFrame)
```

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

In [ ]:

```
InputDataFrame = pd.read_csv('/content/drive/MyDrive/Applied Natural Language Processing/Homework1/Dataset_Input_Data_Frame.csv')
InputDataFrame.head()
# display(InputDataFrame)
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326: DtypeWarning: Columns (0) have mixed types.Specify dtype option on import or set low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Out[ ]:

	star_rating	review_body
0	5	Love this, excellent sun block!!
1	5	The great thing about this cream is that it do...
2	5	Great Product, I'm 65 years old and this is al...
3	5	I use them as shower caps & conditioning caps....
4	5	This is my go-to daily sunblock. It leaves no ...

In [ ]:

```
InputDataFrame.isnull().sum()
```

Out[ ]:

```
star_rating     10
review_body    400
dtype: int64
```

In [ ]:

```
# InputDataFrame.info(verbose = True, show_counts = True)
InputDataFrame.dropna(inplace = True)

InputDataFrame['star_rating'] = InputDataFrame['star_rating'].astype('float').astype('i
nt')
# InputDataFrame.dtypes
```

In [ ]:

```
InputDataFrame.isnull().sum()
```

Out[ ]:

```
star_rating    0
review_body    0
dtype: int64
```

In [ ]:

```
## Reference - https://www.geeksforgeeks.org/create-a-new-column-in-pandas-dataframe-ba
sed-on-the-existing-columns/

def assign_class(value):
    if value == 1 or value == 2:
        return 1
    if value == 3:
        return 2
    if value == 4 or value == 5:
        return 3

InputDataFrame['class'] = InputDataFrame['star_rating'].map(assign_class)

# InputDataFrame['class'].value_counts()
# display(InputDataFrame)
InputDataFrame.head()
```

Out[ ]:

	star_rating	review_body	class
0	5	Love this, excellent sun block!!	3
1	5	The great thing about this cream is that it do...	3
2	5	Great Product, I'm 65 years old and this is al...	3
3	5	I use them as shower caps & conditioning caps....	3
4	5	This is my go-to daily sunblock. It leaves no ...	3

In [ ]:

```
print(average_length(InputDataFrame["review_body"]))
```

```
253.43061308343476
```

In [ ]:

```
InputDataFrame_1 = InputDataFrame.loc[InputDataFrame['class'] == 1].sample(n = 20000)
InputDataFrame_2 = InputDataFrame.loc[InputDataFrame['class'] == 2].sample(n = 20000)
InputDataFrame_3 = InputDataFrame.loc[InputDataFrame['class'] == 3].sample(n = 20000)
```

In [ ]:

```
## Reference - https://pandas.pydata.org/docs/user_guide/merging.html

InputDataFrames = [InputDataFrame_1, InputDataFrame_2, InputDataFrame_3]
InputDataFrameFinal = pd.concat(InputDataFrames)

del InputDataFrame_1, InputDataFrame_2, InputDataFrame_3
InputDataFrameFinal['class'].value_counts()
```

Out[ ]:

```
1    20000
2    20000
3    20000
Name: class, dtype: int64
```

In [ ]:

```
before_cleaning = average_length(InputDataFrameFinal["review_body"])
print("Review Average Length : " + str(before_cleaning))
```

```
Review Average Length : 268.7442
```

Data Cleaning

```
In [ ]:
```

```python
## Reference - https://www.kaggle.com/code/benroshan/sentiment-analysis-amazon-reviews/notebook

class DataCleaning:

  _url = 'https?://\S+|www\.\S+'
  _html = '<[^<]+?>'
  _alphnumeric = '\w*\d\w*'
  _number = '^[\d\s]+'
  _linebresks = '\n'
  _whitespace = ' +'

  def __init__(self, InputDataFrameFinal) -> None:
     self.InputDataFrameFinal = InputDataFrameFinal

  def clean(self):
     print("Data Cleaning Process")

## Reference - https://stackoverflow.com/questions/19790188/expanding-english-language-contractions-in-python
class Contraction(DataCleaning):
  def contraction_function(self, review):

      review = re.sub(r"won\'t", "will not", review)
      review = re.sub(r"can\'t", "can not", review)

      review = re.sub(r"n\'t", " not", review)
      review = re.sub(r"\'re", " are", review)
      review = re.sub(r"\'s", " is", review)
      review = re.sub(r"\'d", " would", review)
      review = re.sub(r"\'ll", " will", review)
      review = re.sub(r"\'t", " not", review)
      review = re.sub(r"\'ve", " have", review)
      review = re.sub(r"\'m", " am", review)
      return review

  def clean(self) -> None:
    InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: self.contraction_function(text))

class Lowercase(DataCleaning):
  def clean(self) -> None:
    InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: str(text).lower())

class RemoveHTMLURL(DataCleaning):
  def html_url(self, review):
    review = re.sub('https?://\S+|www\.\S+', '', review) # html
    review = re.sub('<[^<]+?>', '', review)              # url
    return review

  def clean(self) -> None:
    # InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: re.sub(self._html + '|' + self._url , '', text))
    InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: self.html_url(text))

class RemoveNonAlphabeticalCharacter(DataCleaning):
  def non_alphabetical_function(self, review):
    # review = re.sub('[!\"#\$%&\'\(\)\*\+,-\./:;<=>\?@\[\\\]\^_`{\|}~]', '', review) # punctuation
    # review = re.sub('[^ \w+]', '', review)                                                    # alphnumeric
    # review = re.sub('^[\d\s]+', '', review)                                                   # number
    # review = re.sub('\n', '', review)                                                         # linebreaks
    return review

  def clean(self) -> None:
    InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: re.sub('[^a-zA-Z\s]' , '', text))
    # InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: self.non_alphabetical_function(text))

class RemoveExtraSpaces(DataCleaning):
  def clean(self) -> None:
    InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda text: re.sub(' +', ' ', text))
```

In [ ]:

```
data_cleaning = DataCleaning(InputDataFrameFinal)

contraction = Contraction(InputDataFrameFinal)
contraction.clean()
print("Review Average Length : " + str(average_length(InputDataFrameFinal["review_body"])))
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Review Average Length : 270.02585
268.7442, 270.02585
```

In [ ]:

```
data_cleaning_lowercase = Lowercase(InputDataFrameFinal)
data_cleaning_lowercase.clean()
print("Review Average Length : " + str(average_length(InputDataFrameFinal["review_body"])))
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Review Average Length : 270.02585
268.7442, 270.02585
```

In [ ]:

```
data_cleaning_remove_html_url = RemoveHTMLURL(InputDataFrameFinal)
data_cleaning_remove_html_url.clean()
print("Review Average Length : " + str(average_length(InputDataFrameFinal["review_body"])))
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Review Average Length : 267.65903333333335
268.7442, 267.65903333333335
```

In [ ]:

```
remove_non_alphabetical_character = RemoveNonAlphabeticalCharacter(InputDataFrameFinal)
remove_non_alphabetical_character.clean()
print("Review Average Length : " + str(average_length(InputDataFrameFinal["review_bod
y"])))
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Review Average Length : 259.45501666666667
268.7442, 259.45501666666667
```

In [ ]:

```
remove_white_space = RemoveExtraSpaces(InputDataFrameFinal)
remove_white_space.clean()
print("Review Average Length : " + str(average_length(InputDataFrameFinal["review_bod
y"])))
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Review Average Length : 257.97265
268.7442, 257.97265
```

In [ ]:

```
# display(InputDataFrameFinal)
InputDataFrameFinal.head()
```

Out[ ]:

	star_rating	review_body	class
252877	2	not sure how good the product is but the pump ...	1
2217348	2	smaller than i thought barely covers all my hair	1
1337622	1	all were busted	1
1087229	2	way too watery only at amazon	1
2388595	1	only had this a couple weeks but not worth the...	1

In [ ]:

```
# InputDataFrameFinal = InputDataFrameFinal.sample(frac = 1)
# display(InputDataFrameFinal)
# InputDataFrameFinal.head()
```

In [ ]:

```
print("Average length of reviews before and after data cleaning")
print(average_length_print(before_cleaning, InputDataFrameFinal["review_body"]))
```

```
Average length of reviews before and after data cleaning
268.7442, 257.97265
```

Pre-processing

In [ ]:

```
before_stop_words = average_length(InputDataFrameFinal["review_body"])
print("Review Average Length : " + str(before_stop_words))
```

```
Review Average Length : 257.97265
```

In [ ]:

```
InputDataFrameFinalPreProcessed = InputDataFrameFinal.copy()
```

remove the stop words

In [ ]:

```
## Reference - https://www.geeksforgeeks.org/removing-stop-words-nltk-python/

from nltk.corpus import stopwords
nltk.download('stopwords')
stop_words=stopwords.words('english')
# stop_words = {'a', 'an', 'and', 'are', 'as', 'at', 'be', 'by', 'for', 'from', 'has',
'he', 'in', 'is', 'it', 'its', 'of', 'on', 'that', 'the', 'to', 'was', 'were', 'will',
'with'}

def remove_stopword() -> None:
  InputDataFrameFinalPreProcessed['review_body'] = InputDataFrameFinalPreProcessed['rev
iew_body'].apply(lambda text: ' '.join([word for word in text.split() if word not in (s
top_words)]))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

In [ ]:

```
remove_stopword()
# display(InputDataFrameFinal)
# InputDataFrameFinal.head()

print(average_length_print(before_stop_words, InputDataFrameFinalPreProcessed["review_b
ody"]))
print(average_length_print(before_stop_words, InputDataFrameFinal["review_body"]))
```

```
257.97265, 158.867583333333333
257.97265, 257.97265
```

In [ ]:

```
# import matplotlib.pyplot as plt
# from wordcloud import WordCloud

# consolidated=' '.join(word for word in InputDataFrameFinal['review_body'][InputDataFr
ameFinal['class'] == 3].astype(str))
# wordCloud=WordCloud(width=1600,height=800,random_state=21,max_font_size=110)
# plt.figure(figsize=(15,10))
# plt.imshow(wordCloud.generate(consolidated),interpolation='bilinear')
# plt.axis('off')
# plt.show()
```

perform lemmatization

In [ ]:

```
before_lemmatization = average_length(InputDataFrameFinal["review_body"])
print("Review Average Length : " + str(before_lemmatization))
```

```
Review Average Length : 257.97265
```

In [ ]:

```
InputDataFrameFinal.head()
```

Out[ ]:

	star_rating	review_body	class
252877	2	not sure how good the product is but the pump ...	1
2217348	2	smaller than i thought barely covers all my hair	1
1337622	1	all were busted	1
1087229	2	way too watery only at amazon	1
2388595	1	only had this a couple weeks but not worth the...	1

In [ ]:

```
## Reference - https://www.geeksforgeeks.org/python-lemmatization-with-nltk/

from nltk.stem import WordNetLemmatizer
from nltk.tag import pos_tag
nltk.download('omw-1.4')
nltk.download('averaged_perceptron_tagger')

def lemmatization(review) -> None:
  lemmatizer = WordNetLemmatizer()
  # print(review)
  lemmatized_sentence = []
  for word, tag in pos_tag(review):
    if (review is None):
        return review
    else:
        if tag.startswith('NN'):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'
        lemmatized_sentence.append(lemmatizer.lemmatize(word, pos))
  # print(lemmatized_sentence)
  return lemmatized_sentence

InputDataFrameFinal['review_body'] = InputDataFrameFinal['review_body'].apply(lambda te
xt: ' '.join(lemmatization(text.split()) ))
```

```
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

In [ ]:

```
InputDataFrameFinal.head()
```

Out[ ]:

	star_rating	review_body	class
252877	2	not sure how good the product be but the pump ...	1
2217348	2	small than i think barely cover all my hair	1
1337622	1	all be bust	1
1087229	2	way too watery only at amazon	1
2388595	1	only have this a couple week but not worth the...	1

In [ ]:

```
print("Average length of reviews before and after data preprocessing")
# print(average_length_print(before_lemmatization, InputDataFrameFinal["review_body"]))
print(average_length_print(before_lemmatization, InputDataFrameFinalPreProcessed["review_body"]))
print(average_length_print(before_lemmatization, InputDataFrameFinal["review_body"]))
```

```
Average length of reviews before and after data preprocessing
257.97265, 158.86758333333333
257.97265, 248.36896666666667
```

In [ ]:

```
print("Average length of reviews before and after data preprocessing")
print(average_length_print(before_stop_words, InputDataFrameFinalPreProcessed["review_body"]))
```

```
Average length of reviews before and after data preprocessing
257.97265, 158.86758333333333
```

Rough

TF-IDF Feature Extraction

Using sklearn to extract TF-IDF features with stopwords removal using ngram range of (1,4) and storing the feature vector in X and class output value in Y (**Less Accuracy**)

In [ ]:

```
# max_features=5000, ngram_range=(2,2)

TF_IDF = TfidfVectorizer(ngram_range=(1,4))
X = TF_IDF.fit_transform(InputDataFrameFinalPreProcessed["review_body"])
Y = InputDataFrameFinal['class']
```

In [ ]:

```
x_train ,x_test, y_train, y_test=train_test_split(X, Y, test_size=0.2, random_state=100)
```

In [ ]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(48000, 3237817)
(12000, 3237817)
(48000,)
(12000,)
```

In [ ]:

```
def Evaluation_Metric(y_test,y_pred) -> None:
  precision = precision_score(y_test,y_pred, average=None)
  recall = recall_score(y_test,y_pred, average=None)
  f1 = f1_score(y_test,y_pred, average=None)

  for index in range(3):
    print("{}, {}, {}".format(precision[index], recall[index], f1[index]))
  print("{}, {}, {}".format(precision.mean(), recall.mean(), f1.mean()))
```

Perceptron

In [ ]:

```
# model_perceptron = Perceptron(tol=1e-3, random_state=0)

model_perceptron = Perceptron()
model_perceptron.fit(x_train,y_train)
y_pred = model_perceptron.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
# print(precision_score(y_test,y_pred, average=None))
```

```
0.6515717740162673, 0.7460357412534608, 0.695611358835954
0.6039944903581267, 0.4510156852661353, 0.5164139555424702
0.7156366835276006, 0.7863702271628806, 0.7493379389752447
0.6570676493006649, 0.6611405512274923, 0.6537877511178897
```

SVM

In [ ]:

```
# model_svm = LinearSVC(C=0.5, max_iter=2000)

model_svm = LinearSVC()
model_svm.fit(x_train,y_train)
y_pred = model_svm.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.6916646763792692, 0.7289202114271331, 0.7098039215686274
0.6101973684210527, 0.5723836461815377, 0.5906859493167043
0.7755102040816326, 0.7805703238279362, 0.7780320366132722
0.6924574162939848, 0.693958060478869, 0.6928406358328679
```

Logistic Regression

In [ ]:

```
model_logistic_regression=LogisticRegression(max_iter = 10000)
model_logistic_regression.fit(x_train,y_train)
y_pred = model_logistic_regression.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.6851409460105112, 0.7218726403221747, 0.7030273317808554
0.5957878710987059, 0.6037541784520443, 0.5997445721583653
0.7911179963852311, 0.7404543257612373, 0.7649481962301834
0.6906822711648161, 0.6886937148451521, 0.6892400333898013
```

Naive Bayes

In [ ]:

```
model_naive_bayes = MultinomialNB()
model_naive_bayes.fit(x_train,y_train)
y_pred = model_naive_bayes.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.716324941603945, 0.6946891517744778, 0.7053411704574496
0.5777414075286416, 0.6353818462329648, 0.6051922605927015
0.7943152454780362, 0.7428709521507975, 0.7677322677322678
0.696127198203541, 0.6909806500527468, 0.6927552329274729
```

Final

TF-IDF Feature Extraction

Using sklearn to extract TF-IDF features without stopwords removal using ngram range of (1,4) and storing the feature vector in X and class output value in Y

In [ ]:

```
# max_features=5000, ngram_range=(2,2)

TF_IDF = TfidfVectorizer(ngram_range=(1,4))
X = TF_IDF.fit_transform(InputDataFrameFinal["review_body"])
Y = InputDataFrameFinal['class']
```

In [ ]:

```
x_train ,x_test, y_train, y_test=train_test_split(X, Y, test_size=0.2, random_state=10
0)
```

In [ ]:

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(48000, 4136256)
(12000, 4136256)
(48000,)
(12000,)
```

Helper Function

Helper funtion to print the result matrix in proper format

In [ ]:

```
def Evaluation_Metric(y_test,y_pred) -> None:
  precision = precision_score(y_test,y_pred, average=None)
  recall = recall_score(y_test,y_pred, average=None)
  f1 = f1_score(y_test,y_pred, average=None)

  for index in range(3):
    print("{}, {}, {}".format(precision[index], recall[index], f1[index]))
  print("{}, {}, {}".format(precision.mean(), recall.mean(), f1.mean()))
```

Perceptron

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

In [ ]:

```
# model_perceptron = Perceptron(tol=1e-3, random_state=0)

model_perceptron = Perceptron()
model_perceptron.fit(x_train,y_train)
y_pred = model_perceptron.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
# print(precision_score(y_test,y_pred, average=None))
```

```
0.7233779098563645, 0.7352126856279889, 0.7292472849831482
0.6385936222403925, 0.6024685008999743, 0.6200052924053983
0.7943163289075239, 0.8240695988400193, 0.8089194638832878
0.7187626203347603, 0.7205835951226609, 0.7193906804239448
```

SVM

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

In [ ]:

```
# model_svm = LinearSVC(C=0.5, max_iter=2000)

model_svm = LinearSVC()
model_svm.fit(x_train,y_train)
y_pred = model_svm.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.7561412487205732, 0.7437704505411528, 0.7499048344118767
0.6537467700258398, 0.6505528413473901, 0.6521458950895734
0.8164377072477499, 0.8330111164813919, 0.8246411483253588
0.7421085753313875, 0.7424448027899783, 0.7422306259422697
```

Logistic Regression

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

In [ ]:

```
model_logistic_regression=LogisticRegression(max_iter = 10000)
model_logistic_regression.fit(x_train,y_train)
y_pred = model_logistic_regression.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.7492323439099283, 0.7369745784042285, 0.7430529120669966
0.644544997486174, 0.6592954487014657, 0.65183678657684
0.8128342245989305, 0.8081198646689222, 0.81047018904508
0.7355371886650109, 0.7347966305915388, 0.7351199625629722
```

Naive Bayes

Report Precision, Recall, and f1-score per class and their averages on the testing split of the dataset.

In [ ]:

```
model_naive_bayes = MultinomialNB()
model_naive_bayes.fit(x_train,y_train)
y_pred = model_naive_bayes.predict(x_test)
Evaluation_Metric(y_test,y_pred)

# print(accuracy_score(y_test,y_pred))
# print(metrics.classification_report(y_test,y_pred))
```

```
0.7613969608104505, 0.7188522527057639, 0.739513205592957
0.5889044102973459, 0.7588068912316791, 0.6631460674157303
0.900555898702903, 0.7046882551957467, 0.7906724511930586
0.7502857566035664, 0.7274491330443965, 0.7311105747339153
```