NLP HW1 Report

In order to complete the assignment I have followed the following actions at each step of the assignment.

1. Dataset Preparation
In order to prepare the data before preprocessing first I read columns "review_body", and "star_rating" from the CSV file to store as my data frame(as per the HW guidelines). After this, I separated my data into three classes (1, 2, 3) using a dictionary and added randomly shuffled the data to get different values at each execution. Finally, I picked 20000 rows from each of the three classes to create 60000 data set for further analysis.
Average reviews length before data cleaning: 291.9329333333333

2. Data Cleaning
For data cleaning, I have created a common function that will clean data for the following cases:

- Convert the reviews data into lowercase characters
- Remove numerical characters from the reviews data
- Remove punctuation marks from the reviews data
- Remove extra spaces from the reviews data
- Remove URLs from the reviews data
- Remove HTML tags from the reviews data

After this, I performed contradictions on the review data using a contradiction dictionary to further clean the data.
Average reviews length after data cleaning: 280.53248333333335

3. Data Preprocessing
For data preprocessing, I have used the NLTK package to first remove all the stop words from the dataset. After this, I performed lemmatization using WordNetLemmatizer from the NLTK package.
Average review length after data preprocessing: 174.3622

4. Feature Extraction
After data processing, I used TfidfVectorizer from sklearn package to extract TF-IDF features. I further divided the dataset into 80% training dataset and 20% testing dataset.

5. Results Of The Perceptron Model

	Precision	**Recall**	**f1-score**
Class 1	0.6309553819006806	0.6103389417215314	0.6204759543877045
Class 2	0.4973887092762994	0.5073566717402334	0.5023232450081627
Class 3	0.6622632103688934	0.671468284053576	0.6668339816790061
Average	0.5974024863809645	0.5966666666666667	0.5969493488558317

6. Results Of The SVM Model

	Precision	Recall	f1-score
Class 1	0.6896899420216789	0.6689486552567238	0.6791609780315253
Class 2	0.5369311116637653	0.5825688073394495	0.5588197230490488
Class 3	0.7592223330009971	0.72454804947668886	0.7414800389483934
Average	0.6668724375525644	0.66175	0.66382803145898

7. Results Of The Logistic Regression Model

	Precision	Recall	f1-score
Class 1	0.7070834383665239	0.6853163938431468	0.6960297766749379
Class 2	0.5792091519522506	0.6014979338842975	0.5901431648295958
Class 3	0.7567298105682951	0.7524163568773234	0.7545669193488257
Average	0.6825162612696973	0.6808333333333333	0.6656925870710259

8. Results Of The Naive Bayes Model

	Precision	Recall	f1-score
Class 1	0.6637257373329972	0.6941734774584761	0.6786082474226804
Class 2	0.6187515543397165	0.5682960255824577	0.5924514823193238
Class 3	0.7198404785643071	0.7542439279185166	0.7366407346001785
Average	0.6652229349188391	0.6674166666666667	0.6815468108102689

NLP_HW11

January 25, 2023

Importing All Required Libraries

[1]:
```
import pandas as pd
import numpy as np
import random
from tqdm import tqdm
import nltk
```

[2]:
```
cd /content/drive/MyDrive/NLP
```

```
/content/drive/MyDrive/NLP
```

[ ]:
```
!wget https://s3.amazonaws.com/amazon-reviews-pds/tsv/
↪amazon_reviews_us_Beauty_v1_00.tsv.gz
```

```
--2023-01-25 03:40:38--  https://s3.amazonaws.com/amazon-reviews-
pds/tsv/amazon_reviews_us_Beauty_v1_00.tsv.gz
Resolving s3.amazonaws.com (s3.amazonaws.com)… 52.216.62.40, 52.217.132.0,
52.216.39.72, …
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.62.40|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 914070021 (872M) [application/x-gzip]
Saving to: 'amazon_reviews_us_Beauty_v1_00.tsv.gz.1'

amazon_reviews_us_B 100%[===================>] 871.72M  54.1MB/s    in 22s

2023-01-25 03:41:00 (40.1 MB/s) - 'amazon_reviews_us_Beauty_v1_00.tsv.gz.1'
saved [914070021/914070021]
```

Unzipping Reviews Data

[ ]:
```
!gunzip amazon_reviews_us_Beauty_v1_00.tsv.gz
```

```
gzip: amazon_reviews_us_Beauty_v1_00.tsv already exists; do you wish to
overwrite (y or n)? y
```

Reading Data From CSV File

[3]:
```
dataframe = pd.read_csv(r"amazon_reviews_us_Beauty_v1_00.tsv",sep="\t",usecols␣
↪= ["review_body","star_rating"])
```

```
/usr/local/lib/python3.8/dist-packages/IPython/core/interactiveshell.py:3326:
DtypeWarning: Columns (7) have mixed types.Specify dtype option on import or set
low_memory=False.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Separating Ratings Into Three Classes

[4]:
```
ratingDict = {1:1, 2:1, 3:2, 4:3, 5:3}
filteredDataFrame = dataframe.replace({"star_rating": ratingDict})
filteredDataFrame = filteredDataFrame.sample(frac=1, random_state=14).
↪reset_index(drop=True)
filteredDataFrame
```

[4]:

	star_rating	review_body
0	3	There aren't nearly 600 pieces, but for the pr…
1	3	good stuff, good price - must use with its own…
2	2	I like the feel of the lipstick, but it's too …
3	3	Ive been using this system fairly regularly fo…
4	3	I had a Remington makeup mirror for about a de…
…	…	…
5094558	2	Dont work for my stinky butt
5094559	3	I just Love this colors! They is pretty and tr…
5094560	3	these nail art bows are beautiful bows n pearl…
5094561	5	Leaves my tangled messy long hair soft and smo…
5094562	5	I LOVE this cream!! Makes my hands and entire …

[5094563 rows x 2 columns]

Contradictions Dictionary For Performing Contradictions On Reviews

[5]:
```
contractions_dict = {
"ain't": "are not",
"aren't": "are not",
"can't": "cannot",
"can't've": "cannot have",
"'cause": "because",
"could've": "could have",
"couldn't": "could not",
"couldn't've": "could not have",
"didn't": "did not",
"doesn't": "does not",
"don't": "do not",
"hadn't": "had not",
"hadn't've": "had not have",
"hasn't": "has not",
```

```
"haven't": "have not",
"he'd": "he would",
"he'd've": "he would have",
"he'll": "he will",
"he'll've": "he will have",
"he's": "he is",
"how'd": "how did",
"how'd'y": "how do you",
"how'll": "how will",
"how's": "how is",
"I'd": "I had",
"I'd've": "I would have",
"I'll": "I will",
"I'll've": "I will have",
"I'm": "I am",
"I've": "I have",
"isn't": "is not",
"it'd": "it had",
"it'd've": "it would have",
"it'll": "it will",
"it'll've": "it will have",
"it's": "it is",
"let's": "let us",
"ma'am": "madam",
"mayn't": "may not",
"might've": "might have",
"mightn't": "might not",
"mightn't've": "might not have",
"must've": "must have",
"mustn't": "must not",
"mustn't've": "must not have",
"needn't": "need not",
"needn't've": "need not have",
"o'clock": "of the clock",
"oughtn't": "ought not",
"oughtn't've": "ought not have",
"shan't": "shall not",
"sha'n't": "shall not",
"shan't've": "shall not have",
"she'd": "she would",
"she'd've": "she would have",
"she'll": "she will",
"she'll've": "she will have",
"she's": "she is",
"should've": "should have",
"shouldn't": "should not",
"shouldn't've": "should not have",
```

```
"so've": "so have",
"so's": "so is",
"that'd": "that would",
"that'd've": "that would have",
"that's": "that is",
"there'd": "there would",
"there'd've": "there would have",
"there's": "there is",
"they'd": "they had",
"they'd've": "they would have",
"they'll": "they will",
"they'll've": "they will have",
"they're": "they are",
"they've": "they have",
"to've": "to have",
"wasn't": "was not",
"we'd": "we would",
"we'd've": "we would have",
"we'll": "we will",
"we'll've": "we will have",
"we're": "we are",
"we've": "we have",
"weren't": "were not",
"what'll": "what will",
"what'll've": "what will have",
"what're": "what are",
"what's": "what is",
"what've": "what have",
"when's": "when is",
"when've": "when have",
"where'd": "where did",
"where's": "where is",
"where've": "where have",
"who'll": "who will",
"who'll've": "who will have",
"who's": "who is",
"who've": "who have",
"why's": "why is",
"why've": "why have",
"will've": "will have",
"won't": "will not",
"won't've": "will not have",
"would've": "would have",
"wouldn't": "would not",
"wouldn't've": "would not have",
"y'all": "you all",
"y'all'd": "you all would",
```

```
"y'all'd've": "you all would have",
"y'all're": "you all are",
"y'all've": "you all have",
"you'd": "you would",
"you'd've": "you would have",
"you'll": "you will",
"you'll've": "you will have",
"you're": "you are",
"you've": "you have",
"i'd": "i would",
"i'd've": "i would have",
"i'll": "i will",
"i'll've": "i will have",
"i've": "i have"
}
```

Taking 20000 Data From Each Rating Class

[6]:
```
rating1Data = filteredDataFrame[filteredDataFrame['star_rating']==1].head(20000)
rating2Data = filteredDataFrame[filteredDataFrame['star_rating']==2].head(20000)
rating3Data = filteredDataFrame[filteredDataFrame['star_rating']==3].head(20000)

finalRatingsData = rating1Data.append(rating2Data).append(rating3Data).
↪reset_index()
finalRatingsData
```

[6]:
```
       index star_rating                                        review_body
0          6           1  While this cap fits really well, it smells hor…
1         18           1  I do not like it no is do not do nothing do no…
2         24           1  Bought this product a few months ago. Not happ…
3         37           1  Simple…right? Wrong. It's simple if it actua…
4         49           1  This candle has a nice container, lid, etc but…
…          …           …                                                …
59995  31882           3  I purchased this product because it is suppose…
59996  31885           3  It arrived all in one piece and it smells grea…
59997  31886           3  it goes on smoothly, spreads well and you do n…
59998  31888           3                             Love opi products!!!!
59999  31889           3  Works great! Makes my beard shine (my wife say…

[60000 rows x 3 columns]
```

[7]:
```
finalRatingsData['review_body'] = finalRatingsData["review_body"].apply(str)␣
↪#converts review_body column to string type
```

[8]:
```
averageStringLenBeforeDataCleaning, stringLength = 0, 0
for ratings in finalRatingsData['review_body']:
  stringLength += len(ratings)
```

```
averageStringLenBeforeDataCleaning = stringLength /␣
 ↪len(finalRatingsData['review_body'])
print("Average reviews length before data cleaning : ",␣
 ↪averageStringLenBeforeDataCleaning)
```

```
Average reviews length before data cleaning :  291.9329333333333
```

Performing Preprocessing/Cleaning Of Review Data

[9]:
```
import re

def cleanReviewData(column):
  column = column.lower() #converts string to lowercase
  column = re.sub(r'\d+','',column) #remove numerical characters from the string
  column = re.sub(r'[^\w\s]','', column) #removes punctuations from the string
  column = column.strip() #remove extra spaces from the string
  column = re.sub(r"http\S+", "", column) #removes URLs from the string
  colummn = re.sub(re.compile('<.*?>'), '', column) #removes HTML tags from the␣
 ↪string

  return column


finalRatingsData['review_body'] = finalRatingsData["review_body"].
 ↪apply(cleanReviewData)
finalRatingsData
```

[9]:
```
       index star_rating                                        review_body
0          6           1  while this cap fits really well it smells horr…
1         18           1  i do not like it no is do not do nothing do no…
2         24           1  bought this product a few months ago not happy…
3         37           1  simpleright wrong its simple if it actually wo…
4         49           1  this candle has a nice container lid etc but h…
…          …           …                                                …
59995  31882           3  i purchased this product because it is suppose…
59996  31885           3  it arrived all in one piece and it smells grea…
59997  31886           3  it goes on smoothly spreads well and you do no…
59998  31888           3                                love opi products
59999  31889           3  works great makes my beard shine my wife says …

[60000 rows x 3 columns]
```

Using Contradictions Dictionary To Perform Contradictions

[10]:
```
for reviews in finalRatingsData['review_body']:
  review = reviews.split()
  for char in review:
```

```
    if char in contractions_dict:
      reviews = reviews.replace(char, contractions_dict[char])

finalRatingsData
```

[10]:

```
       index star_rating                                        review_body
0          6           1  while this cap fits really well it smells horr…
1         18           1  i do not like it no is do not do nothing do no…
2         24           1  bought this product a few months ago not happy…
3         37           1  simpleright wrong its simple if it actually wo…
4         49           1  this candle has a nice container lid etc but h…
…          …           …                                                …
59995  31882           3  i purchased this product because it is suppose…
59996  31885           3  it arrived all in one piece and it smells grea…
59997  31886           3  it goes on smoothly spreads well and you do no…
59998  31888           3                                love opi products
59999  31889           3  works great makes my beard shine my wife says …

[60000 rows x 3 columns]
```

[11]:

```
averageStringLengAfterDataCleaning, stringLength = 0, 0
for ratings in finalRatingsData['review_body']:
  stringLength += len(ratings)

averageStringLengAfterDataCleaning = stringLength /␣
 ↪len(finalRatingsData['review_body'])
print("Average reviews length after data cleaning : ",␣
 ↪averageStringLengAfterDataCleaning)
```

```
Average reviews length after data cleaning :  280.53248333333335
```

[12]:

```
averageStringLengBeforeDataPreprocessing, stringLength = 0, 0
for ratings in finalRatingsData['review_body']:
  stringLength += len(ratings)

averageStringLengBeforeDataPreprocessing = stringLength /␣
 ↪len(finalRatingsData['review_body'])
print("Average reviews length before data preprocessing : ",␣
 ↪averageStringLengBeforeDataPreprocessing)
```

```
Average reviews length before data preprocessing :  280.53248333333335
```

Using NLTK Library To Remove Stop Words

[13]:

```
nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))
```

```
finalRatingsData['review_body'] = finalRatingsData['review_body'].apply(lambda␣
↪key: ' '.join([word for word in key.split() if word not in (stop_words)]))
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

Using NLTK Library To Perform Lemmatization

[14]:
```
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')
nltk.download('omw-1.4')
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)])
finalRatingsData['review_body'] = finalRatingsData['review_body'].
↪apply(lemmatize_text)
```

```
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data…
[nltk_data]   Package omw-1.4 is already up-to-date!
```

[15]:
```
averageStringLengAfterDataPreprocessing, stringLength = 0, 0
for ratings in finalRatingsData['review_body'].to_list():
  stringLength += len(ratings)

averageStringLengAfterDataPreprocessing = stringLength /␣
↪len(finalRatingsData['review_body'])
print("Average reviews length after data preprocessing : ",␣
↪averageStringLengAfterDataPreprocessing)
```

```
Average reviews length after data preprocessing :  174.3622
```

Feature Extraction Using TF-IDF

[16]:
```
print(finalRatingsData)
from sklearn.feature_extraction.text import TfidfVectorizer
tfidvectorizer = TfidfVectorizer()
x = tfidvectorizer.fit_transform(finalRatingsData['review_body'])
x
```

```
       index  star_rating                                        review_body
0          6            1  cap fit really well smell horrible tried washi…
1         18            1      like nothing west money personal opinion sory
2         24            1  bought product month ago happy result image wo…
3         37            1  simpleright wrong simple actually work sorry d…
4         49            1  candle nice container lid etc scent little bur…
```

```
…       …          …                                                   …
59995   31882          3   purchased product supposed help hair loss im sure
59996   31885          3   arrived one piece smell great like always husb…
59997   31886          3   go smoothly spread well need much cover face n…
59998   31888          3                                  love opi product
59999   31889          3     work great make beard shine wife say smell good

[60000 rows x 3 columns]
```

[16]:
```
<60000x47252 sparse matrix of type '<class 'numpy.float64'>'
        with 1374165 stored elements in Compressed Sparse Row format>
```

Splitting Data Into Traing And Testing Set

[17]:
```python
from sklearn.model_selection import train_test_split
part1 = finalRatingsData['review_body'].to_list()
part2 = finalRatingsData['star_rating'].to_list()
part1_train, part1_test, part2_train, part2_test = train_test_split(part1,
 ↪part2, test_size=0.2, shuffle = True)
```

[18]:
```python
Train_X_Tfidf = tfidvectorizer.transform(part1_train)
Test_X_Tfidf = tfidvectorizer.transform(part1_test)
```

Running Naive Bayes Model

[19]:
```python
from sklearn import model_selection, naive_bayes, svm
from sklearn.metrics import accuracy_score

Naive = naive_bayes.MultinomialNB()
Naive.fit(Train_X_Tfidf,part2_train)
predictions_NB = Naive.predict(Test_X_Tfidf)
print("Naive Bayes Accuracy Score -> ",accuracy_score(predictions_NB,
 ↪part2_test)*100)
```

```
Naive Bayes Accuracy Score ->  66.74166666666666
```

Running SVM Model

[20]:
```python
from sklearn.svm import LinearSVC

svm = LinearSVC()
svm.fit(Train_X_Tfidf,part2_train)
predictions_svm = svm.predict(Test_X_Tfidf)
print("SVM Accuracy Score -> ",accuracy_score(predictions_svm, part2_test)*100)
```

```
SVM Accuracy Score ->  66.175
```

Running Perceptron Model

[21]:
```
from sklearn.linear_model import Perceptron

perceptron = Perceptron()
perceptron.fit(Train_X_Tfidf,part2_train)
predictions_perp = perceptron.predict(Test_X_Tfidf)
print("Perceptron Accuracy Score -> ",accuracy_score(predictions_perp,␣
↪part2_test)*100)
```

```
Perceptron Accuracy Score ->  59.66666666666667
```

Running Logistic Regression Model

[22]:
```
from sklearn.linear_model import LogisticRegression

logisticRegression = LogisticRegression(max_iter=1000)
logisticRegression.fit(Train_X_Tfidf,part2_train)
predictions_log = logisticRegression.predict(Test_X_Tfidf)
print("LogisticRegression Accuracy Score -> ",accuracy_score(predictions_log,␣
↪part2_test)*100)
```

```
LogisticRegression Accuracy Score ->  68.08333333333333
```

[23]:
```
from sklearn import metrics

predictions_NB_output = metrics.classification_report(predictions_NB,␣
↪part2_test, output_dict=True)
print("Naive Bayes Output -> ",predictions_NB_output)
```

```
Naive Bayes Output ->  {'1': {'precision': 0.6637257373329972, 'recall':
0.6941734774584761, 'f1-score': 0.6786082474226804, 'support': 3793}, '2':
{'precision': 0.6187515543397165, 'recall': 0.5682960255824577, 'f1-score':
0.5924514823193238, 'support': 4378}, '3': {'precision': 0.7198404785643071,
'recall': 0.7542439279185166, 'f1-score': 0.7366407346001785, 'support': 3829},
'accuracy': 0.6674166666666667, 'macro avg': {'precision': 0.6674392567456735,
'recall': 0.6722378103198169, 'f1-score': 0.6692334881140609, 'support': 12000},
'weighted avg': {'precision': 0.6652229349188391, 'recall': 0.6674166666666667,
'f1-score': 0.6656925870710259, 'support': 12000}}
```

Output Of Naive Bayes Model

[24]:
```
for key, val in predictions_NB_output.items():
    if key == "accuracy" or key == "macro avg":
        continue
    if key == "1":
        print("Class 1 Precison : ", val['precision'])
        print("Class 1 Recall : ", val['recall'])
        print("Class 1 f1-score : ", val['f1-score'])
    elif key == "2":
        print("Class 2 Precison : ", val['precision'])
```

```
        print("Class 2 Recall : ", val['recall'])
        print("Class 2 f1-score : ", val['f1-score'])
    elif key == "3":
        print("Class 3 Precison : ", val['precision'])
        print("Class 3 Recall : ", val['recall'])
        print("Class 3 f1-score : ", val['f1-score'])
    elif key == "weighted avg":
        print("Average Precison : ", val['precision'])
        print("Average Recall : ", val['recall'])
        print("Average f1-score : ", val['f1-score'])
```

```
Class 1 Precison :  0.6637257373329972
Class 1 Recall :  0.6941734774584761
Class 1 f1-score :  0.6786082474226804
Class 2 Precison :  0.6187515543397165
Class 2 Recall :  0.5682960255824577
Class 2 f1-score :  0.5924514823193238
Class 3 Precison :  0.7198404785643071
Class 3 Recall :  0.7542439279185166
Class 3 f1-score :  0.7366407346001785
Average Precison :  0.6652229349188391
Average Recall :  0.6674166666666667
Average f1-score :  0.6656925870710259
```

[25]:

```
from sklearn import metrics

predictions_svm_output = metrics.classification_report(predictions_svm,
 ↪part2_test, output_dict=True)
print("SVM Output -> ",predictions_svm_output)
```

```
SVM Output ->  {'1': {'precision': 0.6896899420216789, 'recall':
0.6689486552567238, 'f1-score': 0.6791609780315253, 'support': 4090}, '2':
{'precision': 0.5369311116637653, 'recall': 0.5825688073394495, 'f1-score':
0.5588197230490488, 'support': 3706}, '3': {'precision': 0.7592223330009971,
'recall': 0.7245480494766888, 'f1-score': 0.7414800389483934, 'support': 4204},
'accuracy': 0.66175, 'macro avg': {'precision': 0.6619477955621471, 'recall':
0.6586885040242874, 'f1-score': 0.6598202466763224, 'support': 12000}, 'weighted
avg': {'precision': 0.6668724375525644, 'recall': 0.66175, 'f1-score':
0.66382803145898, 'support': 12000}}
```

Output Of SVM Model

[26]:

```
for key, val in predictions_svm_output.items():
    if key == "accuracy" or key == "macro avg":
        continue
    if key == "1":
        print("Class 1 Precison : ", val['precision'])
        print("Class 1 Recall : ", val['recall'])
```

```
        print("Class 1 f1-score : ", val['f1-score'])
    elif key == "2":
        print("Class 2 Precison : ", val['precision'])
        print("Class 2 Recall : ", val['recall'])
        print("Class 2 f1-score : ", val['f1-score'])
    elif key == "3":
        print("Class 3 Precison : ", val['precision'])
        print("Class 3 Recall : ", val['recall'])
        print("Class 3 f1-score : ", val['f1-score'])
    elif key == "weighted avg":
        print("Average Precison : ", val['precision'])
        print("Average Recall : ", val['recall'])
        print("Average f1-score : ", val['f1-score'])
```

```
Class 1 Precison :  0.6896899420216789
Class 1 Recall :  0.6689486552567238
Class 1 f1-score :  0.6791609780315253
Class 2 Precison :  0.5369311116637653
Class 2 Recall :  0.5825688073394495
Class 2 f1-score :  0.5588197230490488
Class 3 Precison :  0.7592223330009971
Class 3 Recall :  0.7245480494766888
Class 3 f1-score :  0.7414800389483934
Average Precison :  0.6668724375525644
Average Recall :  0.66175
Average f1-score :  0.66382803145898
```

[27]:
```
from sklearn import metrics

predictions_perp_output = metrics.classification_report(predictions_perp,␣
↪part2_test, output_dict=True)
print("Perceptron Output -> ",predictions_perp_output)
```


```
Perceptron Output ->  {'1': {'precision': 0.6309553819006806, 'recall':
0.6103389417215314, 'f1-score': 0.6204759543877045, 'support': 4101}, '2':
{'precision': 0.4973887092762994, 'recall': 0.5073566717402334, 'f1-score':
0.5023232450081627, 'support': 3942}, '3': {'precision': 0.6622632103688934,
'recall': 0.671468284053576, 'f1-score': 0.6668339816790061, 'support': 3957},
'accuracy': 0.5966666666666667, 'macro avg': {'precision': 0.5968691005152911,
'recall': 0.5963879658384469, 'f1-score': 0.5965443936916245, 'support': 12000},
'weighted avg': {'precision': 0.5974024863809645, 'recall': 0.5966666666666667,
'f1-score': 0.5969493488558317, 'support': 12000}}
```


Output Of Perceptron Model

[28]:
```
for key, val in predictions_perp_output.items():
    if key == "accuracy" or key == "macro avg":
        continue
```

```
    if key == "1":
        print("Class 1 Precison : ", val['precision'])
        print("Class 1 Recall : ", val['recall'])
        print("Class 1 f1-score : ", val['f1-score'])
    elif key == "2":
        print("Class 2 Precison : ", val['precision'])
        print("Class 2 Recall : ", val['recall'])
        print("Class 2 f1-score : ", val['f1-score'])
    elif key == "3":
        print("Class 3 Precison : ", val['precision'])
        print("Class 3 Recall : ", val['recall'])
        print("Class 3 f1-score : ", val['f1-score'])
    elif key == "weighted avg":
        print("Average Precison : ", val['precision'])
        print("Average Recall : ", val['recall'])
        print("Average f1-score : ", val['f1-score'])
```

```
Class 1 Precison :  0.6309553819006806
Class 1 Recall :  0.6103389417215314
Class 1 f1-score :  0.6204759543877045
Class 2 Precison :  0.4973887092762994
Class 2 Recall :  0.5073566717402334
Class 2 f1-score :  0.5023232450081627
Class 3 Precison :  0.6622632103688934
Class 3 Recall :  0.671468284053576
Class 3 f1-score :  0.6668339816790061
Average Precison :  0.5974024863809645
Average Recall :  0.5966666666666667
Average f1-score :  0.5969493488558317
```

[29]:
```
from sklearn import metrics

predictions_log_output = metrics.classification_report(predictions_log,
 ↪part2_test, output_dict=True)
print("LogisticRegression Output -> ",predictions_log_output)
```

```
LogisticRegression Output ->  {'1': {'precision': 0.7070834383665239, 'recall':
0.6853163938431468, 'f1-score': 0.6960297766749379, 'support': 4093}, '2':
{'precision': 0.5792091519522506, 'recall': 0.6014979338842975, 'f1-score':
0.5901431648295958, 'support': 3872}, '3': {'precision': 0.7567298105682951,
'recall': 0.7524163568773234, 'f1-score': 0.7545669193488257, 'support': 4035},
'accuracy': 0.6808333333333333, 'macro avg': {'precision': 0.6810074669623566,
'recall': 0.6797435615349227, 'f1-score': 0.6802466202844532, 'support': 12000},
'weighted avg': {'precision': 0.6825162612696973, 'recall': 0.6808333333333333,
'f1-score': 0.6815468108102689, 'support': 12000}}
```

Output Of Logistic Regression Model

[30]:
```
for key, val in predictions_log_output.items():
    if key == "accuracy" or key == "macro avg":
        continue
    if key == "1":
        print("Class 1 Precison : ", val['precision'])
        print("Class 1 Recall : ", val['recall'])
        print("Class 1 f1-score : ", val['f1-score'])
    elif key == "2":
        print("Class 2 Precison : ", val['precision'])
        print("Class 2 Recall : ", val['recall'])
        print("Class 2 f1-score : ", val['f1-score'])
    elif key == "3":
        print("Class 3 Precison : ", val['precision'])
        print("Class 3 Recall : ", val['recall'])
        print("Class 3 f1-score : ", val['f1-score'])
    elif key == "weighted avg":
        print("Average Precison : ", val['precision'])
        print("Average Recall : ", val['recall'])
        print("Average f1-score : ", val['f1-score'])
```

```
Class 1 Precison :  0.7070834383665239
Class 1 Recall :  0.6853163938431468
Class 1 f1-score :  0.6960297766749379
Class 2 Precison :  0.5792091519522506
Class 2 Recall :  0.6014979338842975
Class 2 f1-score :  0.5901431648295958
Class 3 Precison :  0.7567298105682951
Class 3 Recall :  0.7524163568773234
Class 3 f1-score :  0.7545669193488257
Average Precison :  0.6825162612696973
Average Recall :  0.6808333333333333
Average f1-score :  0.6815468108102689
```