CSCI 544 HW 1.

1. Data Preparation

a. Include 3 sample reviews in your report.

⇨ We use the following code to get this output. After selecting only the star_ratings and review_body columns, we use pandas head(3) method to get first 3 records.

```
⇨ rev_rat = amazon_reviews[['star_rating','review_body']]
⇨  rev_rat.head(3)
```

	star_rating	review_body
0	5.0	Beautiful. Looks great on counter.
1	5.0	I personally have 5 days sets and have also bo...
2	5.0	Fabulous and worth every penny. Used for clean...

b. Statistics of star ratings.

⇨ We can get the counts of each star by selecting the star_rating column and then apply pandas value_counts() method which returns count of unique values in sorted order.

```
⇨ # Statistics of ratings
⇨ rev_rat['star_rating'].value_counts()
```

```
5.0    3124759
4.0     731733
1.0     426900
3.0     349547
2.0     241948
Name: star_rating, dtype: int64
```

c. Statistics of the 3 classes.

⇨ I first dropped all the NaN type ratings so that it can help me in getting the correct classes. Then, I converted star_rating column to int to apply numpy where clause. After that I created a new column ‘class’ in which all ratings above 2 where class 1 and others as 0. Since we counted ratings 3 as class 1 then I changed its rating to class 3 meaning a neutral rating. After all this steps I dropped the class 3 as mentioned in the assignment.

```
⇨ rev_rat['star_rating']=rev_rat['star_rating'].astype(int) # convert values of star_rating to int so that we can use numpy where clause
⇨  rev_rat['class']=np.where(rev_rat['star_rating']<3,0,1) # set class based on the given requirements
⇨  rev_rat['class']=np.where(rev_rat['star_rating']==3,3,rev_rat['class']) # we will now change class of ratings 3 as on previous step we added it to class 1
⇨
```

```
# Statistics of ratings after classes
rev_rat['class'].value_counts()
```

```
1    3856296
0     668809
3     349539
Name: class, dtype: int64
```

2. Data Cleaning

a. Average length of characters in review before cleaning

⇨ I looped over all the reviews and measured the length of characters and saved it under char_len variable. After that I printed the mean of it.

```
#Average char length in review_body before data cleaning
from statistics import mean
char_len=[len(char) for char in rev_rat['review_body']]
print(mean(char_len))
```

```
323.796825
```

b. Average length of characters in review after cleaning

⇨ Used the same function as above.

```
#Average char length in review_body after data cleaning
from statistics import mean
char_len_after=[len(char) for char in rev_rat['review_body']]
print(mean(char_len_after))
```

```
309.058895
```

3. Preprocessing

a. 3 sample reviews before data Cleaning and Preprocessing.

⇨ I used rev_rat.head(3) to print the 3 sample reviews.

	star_rating	review_body	class
8	5	sharp and look great	1
27	5	I've been waiting my whole life for these!	1
64	5	Good water bottle. Water tastes so much bette...	1

b. Average length of reviews before data preprocessing.

⇨ It will be the same as avg length after data cleaning which is this.

```
309.058895
```

c. 3 sample reviews after data Cleaning and Preprocessing.

⇨ I used rev_rat.head(3) to print the 3 sample reviews.

	star_rating	review_body	class
8	5	sharp look great	1
27	5	I waiting whole life	1
64	5	good water bottle water taste much better old ...	1

d. Average length of reviews after data preprocessing.

⇨ The avg length has been reduced drastically after the preprocessing.

```
#Average char length in review_body after data preprocessing
from statistics import mean
char_len_af_pre=[len(char) for char in rev_rat['review_body']]
print(mean(char_len_af_pre))
✓ 0.5s

191.49201
```

4. Perceptron

a. Report Accuracy, Precision, Recall and F1 Score.

⇨ After training the model with 80% training data and trying different hyperparamteres I got an Accuracy of 85%.

```
...   Perceptron Model
      Accuracy of Test:  85.4275 %
      Precision of Test:  85.46517552113897 %
      Recall of Test:  87.06732216313836 %
      F1 Score of Test:  85.62196295108654 %
      Accuracy of Train:  93.16187500000001 %
      Precision of Train:  93.19302715779874 %
      Recall of Train:  94.4920440636475 %
      F1 Score of Train:  93.2568273005738 %
```

5. SVM

a. Report Accuracy, Precision, Recall and F1 Score

⇨ After training the model with 80% training data and trying different hyperparamteres I got an Accuracy of 89%.

```
...   SVM Model
      Accuracy of Test:  89.3475 %
      Precision of Test:  89.34906891308866 %
      Recall of Test:  89.03882813283836 %
      F1 Score of Test:  89.28292965114817 %
      Accuracy of Train:  94.03375 %
      Precision of Train:  94.0338741763167 %
      Recall of Train:  93.93375465241176 %
      F1 Score of Train:  94.03240729164062 %
```

6. Logistic Regression

a. Report Accuracy, Precision, Recall and F1 Score

⇨ After training the model with 80% training data and trying different hyperparamteres I got an Accuracy of 89.5%.

```
Logistic Regression Model
Accuracy of Test:  89.58500000000001 %
Precision of Test:  89.5887996961635 %
Recall of Test:  89.10905989766229 %
F1 Score of Test:  89.50418220296281 %
Accuracy of Train:  91.31937500000001 %
Precision of Train:  91.32108997598142 %
Recall of Train:  90.9874353658232 %
F1 Score of Train:  91.29701921811655 %
```

7. Multinomial Naïve Bayes

a. Report Accuracy, Precision, Recall and F1 Score

⇨ After training the model with 80% training data and trying different hyperparamteres I got an Accuracy of 86.4%.

```
Multinomial Naive Bayes Model
Accuracy of Test:  86.815 %
Precision of Test:  86.83457319324135 %
Recall of Test:  85.66268686665998 %
F1 Score of Test:  86.62304063308477 %
Accuracy of Train:  89.08625 %
Precision of Train:  89.1171864079181 %
Recall of Train:  87.67516798641121 %
F1 Score of Train:  88.93815961180302 %
```

In [2]:

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

```
[nltk_data] Error loading wordnet: <urlopen error [SSL:
[nltk_data]     CERTIFICATE_VERIFY_FAILED] certificate verify failed:
[nltk_data]     unable to get local issuer certificate (_ssl.c:1129)>
```

In [3]:

```
import nltk
import ssl

try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context

nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[3]:

```
True
```

In [ ]:

```
#! pip install bs4 # in case you don't have it installed

# Dataset: https://s3.amazonaws.com/amazon-reviews-pds/tsv/amazon_reviews_us_Kitchen_v1_0
0.tsv.gz
```

Read Data

In [4]:

```
amazon_reviews = pd.read_csv('/Users/shubh1/Downloads/amazon.tsv', sep ='\t', error_bad_
lines=False,warn_bad_lines=False)
```

Keep Reviews and Ratings

In [570]:

```
rev_rat = amazon_reviews[['star_rating','review_body']]
 #rev_rat.head(3)
```

In [573]:

```
# Statistics of ratings
rev_rat['star_rating'].value_counts()
```

Out[573]:

```
5.0    3124759
4.0     731733
1.0     426900
```

```
3.0      349547
2.0      241948
Name: star_rating, dtype: int64
```

In [576]:

```
rev_rat=rev_rat.dropna() # Dropping reviews that have type NaN as this will cause when ra
ndomly selecting classes with a particular class
rev_rat = rev_rat.reset_index(drop=True) # reseting the index tso that it covers up for t
he dropped values
```

Labelling Reviews:

The reviews with rating 4,5 are labelled to be 1 and 1,2 are labelled as 0. Discard the reviews with rating 3'

In [579]:

```
rev_rat['star_rating']=rev_rat['star_rating'].astype(int) # convert values of star_rating
to int so that we can use numpy where clause
rev_rat['class']=np.where(rev_rat['star_rating']<3,0,1) # set class based on the given r
equirements
rev_rat['class']=np.where(rev_rat['star_rating']==3,3,rev_rat['class']) # we will now ch
ange class of ratings 3 as on previous step we added it to class 1

 #rev_rat.head(50)
```

In [594]:

```
# Statistics of ratings after classes
ans = rev_rat['class'].value_counts()
#print(ans)
print('Class 1:',ans[0],', Class 0:',ans[1],', Class Neutral or 3:',ans[3])
```

```
Class 1: 668809 , Class 0: 3856296 , Class Neutral or 3: 349539
```

In [117]:

```
# Discarding reviews with ratings 3 and class 3
rev_rat = rev_rat.loc[rev_rat["class"] != 3]
#rev_rat.head(50)
```

In [119]:

```
#rev_rat['class'].value_counts()
```

Out[119]:

```
1    3856296
0     668809
Name: class, dtype: int64
```

We select 200000 reviews randomly with 100,000 positive and 100,000 negative reviews.

In [121]:

```
class0_random = rev_rat.star_rating[rev_rat['class'].eq(0)].sample(100000).index #random
ly select class 0 sample
class1_random = rev_rat.star_rating[rev_rat['class'].eq(1)].sample(100000).index
rev_rat = rev_rat.loc[class0_random.union(class1_random)] #unify both the dataframes
#display(rev_rat['class'].value_counts())

#rev_rat.head(50)
```

```
1    100000
0    100000
```

```
Name: class, dtype: int64
```

In [123]:

```
# 3 samples before Data Cleaning
rev_rat.head(3)
```

Out[123]:

	star_rating	review_body	class
8	5	sharp and look great	1
27	5	I've been waiting my whole life for these!	1
64	5	Good water bottle. Water tastes so much bette...	1

In [125]:

```
#Average char length in review_body before data cleaning
from statistics import mean
char_len=[len(char) for char in rev_rat['review_body']]
print(mean(char_len))
```

```
323.796825
```

Data Cleaning

Convert the all reviews into the lower case.

In [127]:

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

Out[127]:

	star_rating	review_body	class
8	5	sharp and look great	1
27	5	i've been waiting my whole life for these!	1
64	5	good water bottle. water tastes so much bette...	1
78	5	perfect thickness for my vegetable-prep needs,...	1
115	4	i like the pot very much. it heats very quickl...	1

remove the HTML and URLs from the reviews

In [129]:

```
rev_rat['review_body'] = rev_rat['review_body'].replace(r'<.*?>+', '', regex=True) #remo
ves html tags
rev_rat['review_body'] = rev_rat['review_body'].replace(r'http\S+', '', regex=True).repl
ace(r'www\S+', '', regex=True) #removes url
```

remove non-alphabetical characters

In [131]:

```
rev_rat['review_body'] = rev_rat['review_body'].replace(r'[^a-zA-Z\' ]+', '', regex=True
) #I am not removing apostrophe as they will help in contractions
#rev_rat.head(5)
```

Remove the extra spaces between the words

In [133]:

```
rev_rat['review_body']=rev_rat['review_body'].replace(r' +',' ',regex=True)
```

In [135]:

```
import sys
!{sys.executable} -m pip install contractions
import contractions
```

```
Requirement already satisfied: contractions in /Library/Frameworks/Python.framework/Versi
ons/3.9/lib/python3.9/site-packages (0.0.52)
Requirement already satisfied: textsearch>=0.0.21 in /Library/Frameworks/Python.framework
/Versions/3.9/lib/python3.9/site-packages (from contractions) (0.0.21)
Requirement already satisfied: pyahocorasick in /Library/Frameworks/Python.framework/Vers
ions/3.9/lib/python3.9/site-packages (from textsearch>=0.0.21->contractions) (1.4.2)
Requirement already satisfied: anyascii in /Library/Frameworks/Python.framework/Versions/
3.9/lib/python3.9/site-packages (from textsearch>=0.0.21->contractions) (0.3.0)
WARNING: You are using pip version 21.2.3; however, version 21.2.4 is available.
You should consider upgrading via the '/usr/local/bin/python3 -m pip install --upgrade pi
p' command.
```

perform contractions on the reviews.

In [137]:

```
rev_rat['review_body']= [contractions.fix(words) for words in rev_rat['review_body']]
#rev_rat.head(5)
```

In [139]:

```
#Average char length in review_body after data cleaning
from statistics import mean
char_len_after=[len(char) for char in rev_rat['review_body']]
print(mean(char_len_after))
```

```
309.058895
```

Pre-processing

remove the stop words

In [141]:

```
from nltk.corpus import stopwords
st_words = stopwords.words('english')
rev_rat['review_body'] = rev_rat['review_body'].apply(lambda x: ' '.join([i for i in x.s
plit() if i not in (st_words)]))
```

perform lemmatization

In [146]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
rev_rat['review_body']=rev_rat['review_body'].apply(lambda x: " ".join([lemmatizer.lemma
tize(j) for j in nltk.word_tokenize(x)]))
```

In [148]:

```
# 3 samples after preprocessing
```

```
rev_rat.head(3)
```

Out[148]:

	star_rating	review_body	class
8	5	sharp look great	1
27	5	I waiting whole life	1
64	5	good water bottle water taste much better old ...	1

In [150]:

```
#Average char length in review_body after data preprocessing
from statistics import mean
char_len_af_pre=[len(char) for char in rev_rat['review_body']]
print(mean(char_len_af_pre))
```

```
191.49201
```

In [160]:

```
print('Change in Avg char length from normal to Data Cleaning')
print(str(mean(char_len))+","+str(mean(char_len_after)))
```

```
323.796825,309.058895
```

In [162]:

```
print('Change in Avg char length from Data Cleaning to Data Preprocessing')
print(str(mean(char_len_after))+","+str(mean(char_len_af_pre)))
```

```
309.058895,191.49201
```

In [164]:

```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in /Library/Frameworks/Python.framework/Versi
ons/3.9/lib/python3.9/site-packages (0.24.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /Library/Frameworks/Python.framewo
rk/Versions/3.9/lib/python3.9/site-packages (from scikit-learn) (2.2.0)
Requirement already satisfied: scipy>=0.19.1 in /Library/Frameworks/Python.framework/Vers
ions/3.9/lib/python3.9/site-packages (from scikit-learn) (1.7.1)
Requirement already satisfied: numpy>=1.13.3 in /Library/Frameworks/Python.framework/Vers
ions/3.9/lib/python3.9/site-packages (from scikit-learn) (1.21.2)
Requirement already satisfied: joblib>=0.11 in /Library/Frameworks/Python.framework/Versi
ons/3.9/lib/python3.9/site-packages (from scikit-learn) (1.0.1)
WARNING: You are using pip version 21.2.3; however, version 21.2.4 is available.
You should consider upgrading via the '/usr/local/bin/python3 -m pip install --upgrade pi
p' command.
Note: you may need to restart the kernel to use updated packages.
```

TF-IDF Feature Extraction

In [166]:

```
from sklearn.feature_extraction.text import TfidfVectorizer

vec = TfidfVectorizer()
tf_idf_ft = vec.fit_transform(rev_rat['review_body'])
```

In [169]:

```
from sklearn.model_selection import train_test_split
class_val=rev_rat['class']
amazoz_data = train_test_split(tf_idf_ft,
                               class_val,
                               test_size=0.2)
```

```
train_x, test_x, train_y, test_y = amazoz_data
```

Perceptron

In [309]:

```
from sklearn.linear_model import Perceptron
perc = Perceptron(random_state=53,
               max_iter=100000,
               tol = 0.0001
               )
perc.fit(train_x, train_y)
```

Out[309]:

```
Perceptron(max_iter=100000, random_state=53, tol=0.0001)
```

In [555]:

```
from sklearn.metrics import classification_report

final_report = classification_report(perc.predict(test_x), test_y)
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
print('Perceptron Model')
#print(accuracy_score(test_y,perc.predict(test_x) ) * 100,'%',
#',',precision_score(test_y, perc.predict(test_x), average='macro') *100,'%',
#',',recall_score(test_y,perc.predict(test_x) ) * 100,'%',
#',',f1_score(test_y,perc.predict(test_x) ) * 100,'%',
#',',accuracy_score(train_y,perc.predict(train_x) ) * 100,'%',
#',',precision_score(train_y, perc.predict(train_x), average='macro') *100,'%',
#',',recall_score(train_y,perc.predict(train_x) ) * 100,'%',
#',',f1_score(train_y,perc.predict(train_x) ) * 100,'%')

print('Accuracy of Test: ', accuracy_score(test_y,perc.predict(test_x) ) * 100,'%')
print('Precision of Test: ',precision_score(test_y, perc.predict(test_x), average='macro
') *100,'%')
print('Recall of Test: ', recall_score(test_y,perc.predict(test_x) ) * 100,'%')
print('F1 Score of Test: ', f1_score(test_y,perc.predict(test_x) ) * 100,'%')

print('Accuracy of Train: ', accuracy_score(train_y,perc.predict(train_x) ) * 100,'%')
print('Precision of Train: ',precision_score(train_y, perc.predict(train_x), average='ma
cro') *100,'%')
print('Recall of Train: ', recall_score(train_y,perc.predict(train_x) ) * 100,'%')
print('F1 Score of Train: ', f1_score(train_y,perc.predict(train_x) ) * 100,'%')
```

```
Perceptron Model
Accuracy of Test:  85.4275 %
Precision of Test:  85.46517552113897 %
Recall of Test:  87.06732216313836 %
F1 Score of Test:  85.62196295108654 %
Accuracy of Train:  93.16187500000001 %
Precision of Train:  93.19302715779874 %
Recall of Train:  94.4920440636475 %
F1 Score of Train:  93.2568273005738 %
```

SVM

In [359]:

```
from sklearn.svm import LinearSVC
svm_model = LinearSVC(random_state=8)
svm_model.fit(train_x, train_y)
```

Out[359]:

```
LinearSVC(random_state=8)
```

In [558]:

```
print('SVM Model')
#print(accuracy_score(test_y,svm_model.predict(test_x) ) * 100,'%',
#',',precision_score(test_y, svm_model.predict(test_x), average='macro') *100,'%',
#',',recall_score(test_y,svm_model.predict(test_x) ) * 100,'%',
#',',f1_score(test_y,svm_model.predict(test_x) ) * 100,'%',
#',',accuracy_score(train_y,svm_model.predict(train_x) ) * 100,'%',
#',',precision_score(train_y, svm_model.predict(train_x), average='macro') *100,'%',
#',',recall_score(train_y,svm_model.predict(train_x) ) * 100,'%',
#',',f1_score(train_y,svm_model.predict(train_x) ) * 100,'%')

print('Accuracy of Test: ', accuracy_score(test_y,svm_model.predict(test_x) ) * 100,'%')
print('Precision of Test: ',precision_score(test_y, svm_model.predict(test_x), average='
macro') *100,'%')
print('Recall of Test: ', recall_score(test_y,svm_model.predict(test_x) ) * 100,'%')
print('F1 Score of Test: ', f1_score(test_y,svm_model.predict(test_x) ) * 100,'%')

print('Accuracy of Train: ', accuracy_score(train_y,svm_model.predict(train_x) ) * 100,'
%')
print('Precision of Train: ',precision_score(train_y, svm_model.predict(train_x), averag
e='macro') *100,'%')
print('Recall of Train: ', recall_score(train_y,svm_model.predict(train_x) ) * 100,'%')
print('F1 Score of Train: ', f1_score(train_y,svm_model.predict(train_x) ) * 100,'%')
```

```
SVM Model
Accuracy of Test:  89.3475 %
Precision of Test:  89.34906891308866 %
Recall of Test:  89.03882813283836 %
F1 Score of Test:  89.28292965114817 %
Accuracy of Train:  94.03375 %
Precision of Train:  94.0338741763167 %
Recall of Train:  93.93375465241176 %
F1 Score of Train:  94.03240729164062 %
```

Logistic Regression

In [478]:

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(max_iter=10000,solver='saga',tol=0.0001)
log_reg.fit(train_x, train_y)
```

Out[478]:

```
LogisticRegression(max_iter=10000, solver='saga')
```

In [561]:

```
print('Logistic Regression Model')
#print(accuracy_score(test_y,log_reg.predict(test_x) ) * 100,'%',
#',',precision_score(test_y, log_reg.predict(test_x), average='macro') *100,'%',
#',',recall_score(test_y,log_reg.predict(test_x) ) * 100,'%',
#',',f1_score(test_y,log_reg.predict(test_x) ) * 100,'%',
#',',accuracy_score(train_y,log_reg.predict(train_x) ) * 100,'%',
#',',precision_score(train_y, log_reg.predict(train_x), average='macro') *100,'%',
#',',recall_score(train_y,log_reg.predict(train_x) ) * 100,'%',
#',',f1_score(train_y,log_reg.predict(train_x) ) * 100,'%')


print('Accuracy of Test: ', accuracy_score(test_y,log_reg.predict(test_x) ) * 100,'%')
print('Precision of Test: ',precision_score(test_y, log_reg.predict(test_x), average='ma
cro') *100,'%')
print('Recall of Test: ', recall_score(test_y,log_reg.predict(test_x) ) * 100,'%')
print('F1 Score of Test: ', f1_score(test_y,log_reg.predict(test_x) ) * 100,'%')

print('Accuracy of Train: ', accuracy_score(train_y,log_reg.predict(train_x) ) * 100,'%'
)
```

```
print('Precision of Train: ',precision_score(train_y, log_reg.predict(train_x), average=
'macro') *100,'%')
print('Recall of Train: ', recall_score(train_y,log_reg.predict(train_x) ) * 100,'%')
print('F1 Score of Train: ', f1_score(train_y,log_reg.predict(train_x) ) * 100,'%')
```

```
Logistic Regression Model
Accuracy of Test:  89.58500000000001 %
Precision of Test:  89.5887996961635 %
Recall of Test:  89.10905989766229 %
F1 Score of Test:  89.50418220296281 %
Accuracy of Train:  91.31937500000001 %
Precision of Train:  91.32108997598142 %
Recall of Train:  90.9874353658232 %
F1 Score of Train:  91.29701921811655 %
```

Naive Bayes

In [486]:

```
from sklearn.naive_bayes import MultinomialNB
multi_nb = MultinomialNB()
multi_nb.fit(train_x, train_y)
```

Out[486]:

```
MultinomialNB()
```

In [564]:

```
print('Multinomial Naive Bayes Model')
#print(accuracy_score(test_y,multi_nb.predict(test_x) ) * 100,'%',
#',',precision_score(test_y, multi_nb.predict(test_x), average='macro') *100,'%',
#',',recall_score(test_y,multi_nb.predict(test_x) ) * 100,'%',
#',',f1_score(test_y,multi_nb.predict(test_x) ) * 100,'%',
#',',accuracy_score(train_y,multi_nb.predict(train_x) ) * 100,'%',
#',',precision_score(train_y, multi_nb.predict(train_x), average='macro') *100,'%',
#',',recall_score(train_y,multi_nb.predict(train_x) ) * 100,'%',
#',',f1_score(train_y,multi_nb.predict(train_x) ) * 100,'%')

print('Accuracy of Test: ', accuracy_score(test_y,multi_nb.predict(test_x) ) * 100,'%')
print('Precision of Test: ',precision_score(test_y, multi_nb.predict(test_x), average='m
acro') *100,'%')
print('Recall of Test: ', recall_score(test_y,multi_nb.predict(test_x) ) * 100,'%')
print('F1 Score of Test: ', f1_score(test_y,multi_nb.predict(test_x) ) * 100,'%')

print('Accuracy of Train: ', accuracy_score(train_y,multi_nb.predict(train_x) ) * 100,'%
')
print('Precision of Train: ',precision_score(train_y, multi_nb.predict(train_x), average
='macro') *100,'%')
print('Recall of Train: ', recall_score(train_y,multi_nb.predict(train_x) ) * 100,'%')
print('F1 Score of Train: ', f1_score(train_y,multi_nb.predict(train_x) ) * 100,'%')
```

```
Multinomial Naive Bayes Model
Accuracy of Test:  86.815 %
Precision of Test:  86.83457319324135 %
Recall of Test:  85.66268686665998 %
F1 Score of Test:  86.62304063308477 %
Accuracy of Train:  89.08625 %
Precision of Train:  89.1171864079181 %
Recall of Train:  87.67516798641121 %
F1 Score of Train:  88.93815961180302 %
```