MAJOR PROJECT 2023

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
GURUKULA KANGRI (DEEMED TO BE UNIVERSITY)

MAY 2023

Sentiment Analysis and Affective Computing in Multimedia Data on Social Networks

GitHub Repository link
https://github.com/colonel-chirag/Sentiment-Analysis-and-affective-computing

MAJOR PROJECT 2023

Project Mentor : **Mr. Sumit Bansal** (Assistant Professor, Dept. of CSE, FET,GK(DU))

Project Contributors :

Chirag Patel (Team Leader) | 196301029
Kunal Singh Shekhawat | 196301051
Divyank Singh | 196301036
Ashish Bibyan | 196301016

INDEX

Introduction :

The rise of user-generated data from social networking platforms has led to the **need for analyzing people's opinions, attitudes, emotions,** and evaluations from written language.

Sentiment analysis and affective computing are two fields of study that **aim to understand users' subjective perceptions** and emotions from user-generated data.

This presentation highlights the importance of sentiment analysis and affective computing in **analyzing big data on social networks** and explores recent trends in this area.

What is Sentiment Analysis?

Sentiment analysis involves using computational techniques to **analyze and interpret emotions**, opinions, and attitudes from user-generated data.

Psychological and cognitive models are also used to model sentiments and emotions accurately.

Social computing techniques, such as **social network analysis, user profiling, and mining user reviews**, are incorporated into sentiment analysis to enhance its performance.

What is Affective Computing?

Affective computing is an area of research that involves **developing systems and devices** that can recognize, interpret, process, and simulate human emotions.

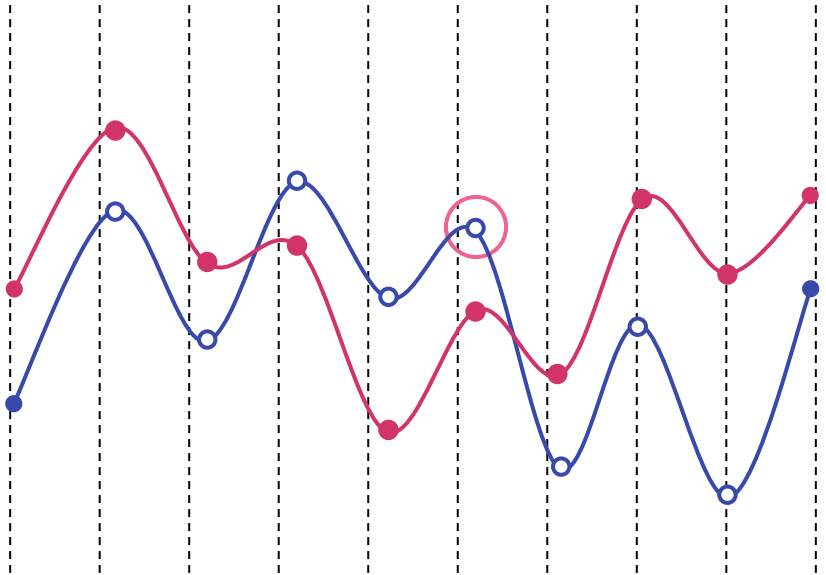
It focuses on creating technology that can **interact with humans** based on their emotional states.

Affective computing aims to make technology more human-like and improve human-computer interaction.

Conclusion

- **Sentiment analysis** and **affective computing** are crucial fields of study that aim to understand users' subjective perceptions and emotions from user-generated data.

- The integration of these fields can facilitate the understanding of big data and **improve the performance** of various social computing applications.



Code/ Output Snippets:

[2]:

```
import re
import numpy as np
import pandas as pd
# plotting
import seaborn as sns
from wordcloud import WordCloud
import matplotlib.pyplot as plt
# nltk
from nltk.stem import WordNetLemmatizer
# sklearn
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.linear_model import SGDClassifier


from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

[4]:

```
df.head()
```

[4]:

	target	ids	date	flag	user	text
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man....
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....

+ Code + Markdown

[5]:

```
df.columns
```

[5]:

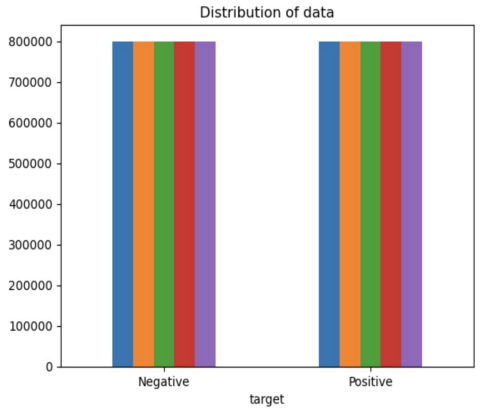
```
Index(['target', 'ids', 'date', 'flag', 'user', 'text'], dtype='object')
```

[6]:

```
print('length of data is', len(df))
```

```
length of data is 1600000
```

[14]:
```
# Plotting the distribution for dataset.
ax = df.groupby('target').count().plot(kind='bar', title='Distribution of data',legend=False)
ax.set_xticklabels(['Negative','Positive'], rotation=0)
# Storing data in lists.
text, sentiment = list(df['text']), list(df['target'])
```



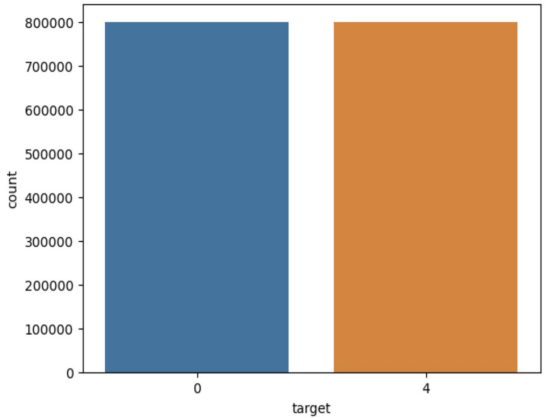
[19]:
```
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
```

[20]:
```
data_pos = data_pos.iloc[:int(500000)]
data_neg = data_neg.iloc[:int(500000)]
```

[21]:
```
dataset = pd.concat([data_pos, data_neg])
```

[15]:
```
import seaborn as sns
sns.countplot(x='target', data=df)
```

[15…] <AxesSubplot:xlabel='target', ylabel='count'>



[23]:
```
stopwordlist = ['a', 'about', 'above', 'after', 'again', 'ain', 'all', 'am', 'an',
             'and','any','are', 'as', 'at', 'be', 'because', 'been', 'before',
             'being', 'below', 'between','both', 'by', 'can', 'd', 'did', 'do',
             'does', 'doing', 'down', 'during', 'each','few', 'for', 'from',
             'further', 'had', 'has', 'have', 'having', 'he', 'her', 'here',
             'hers', 'herself', 'him', 'himself', 'his', 'how', 'i', 'if', 'in',
             'into','is', 'it', 'its', 'itself', 'just', 'll', 'm', 'ma',
             'me', 'more', 'most','my', 'myself', 'now', 'o', 'of', 'on', 'once',
             'only', 'or', 'other', 'our', 'ours','ourselves', 'out', 'own', 're','s', 'same', 'she', "shes", 'should', "shouldve",'so', 'some'
             't', 'than', 'that', "thatll", 'the', 'their', 'theirs', 'them',
             'themselves', 'then', 'there', 'these', 'they', 'this', 'those',
             'through', 'to', 'too','under', 'until', 'up', 've', 'very', 'was',
             'we', 'were', 'what', 'when', 'where','which','while', 'who', 'whom',
             'why', 'will', 'with', 'won', 'y', 'you', "youd","youll", "youre",
             "youve", 'your', 'yours', 'yourself', 'yourselves']
```

+ Code + Markdown

[24]:
```
STOPWORDS = set(stopwordlist)
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])
dataset['text'] = dataset['text'].apply(lambda text: cleaning_stopwords(text))
dataset['text'].head()
```

[24…]
```
800000              love @health4uandpets u guys r best!!
800001    im meeting one besties tonight! cant wait!! - ...
800002    @darealsunisakim thanks twitter add, sunisa! g...
800003    sick really cheap hurts much eat real food plu...
800004                 @lovesbrooklyn2 effect everyone
Name: text, dtype: object
```

[25]:
```
import string
english_punctuations = string.punctuation
punctuations_list = english_punctuations
def cleaning_punctuations(text):
    translator = str.maketrans('', '', punctuations_list)
    return text.translate(translator)
dataset['text']= dataset['text'].apply(lambda x: cleaning_punctuations(x))
dataset['text'].tail()
```

[25…]
```
499995    sleslicountmein wtf amp ive always wanted get ...
499996    heyitsmade  no idea im using web p tweetdeck w...
499997    ww didnt worki gained back probably put chips ...
499998                 dolphinnancy sorry mommy miss too
499999    terrible headache last night weight decreased ...
Name: text, dtype: object
```

+ Code + Markdown

[26]:
```
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
dataset['text'].tail()
```

[26…]
```
499995    sleslicountmein wtf amp ive always wanted get ...
499996    heyitsmade  no idea im using web p tweetdeck w...
499997    ww didnt worki gained back probably put chips ...
499998                 dolphinnancy sorry mommy miss too
499999    terrible headache last night weight decreased ...
Name: text, dtype: object
```

[*]:
```
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_URLs(x))
dataset['text'].tail()
```

[27…]
```
499995    sleslicountmein wtf amp ive always wanted get ...
499996    heyitsmade  no idea im using web p tweetdeck w...
499997    ww didnt worki gained back probably put chips ...
499998                 dolphinnancy sorry mommy miss too
499999    terrible headache last night weight decreased ...
Name: text, dtype: object
```

[28]:
```
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)
dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
dataset['text'].tail()
```

[28…]
```
499995    sleslicountmein wtf amp ive always wanted get ...
499996    heyitsmade  no idea im using web p tweetdeck w...
499997    ww didnt worki gained back probably put chips ...
499998                 dolphinnancy sorry mommy miss too
499999    terrible headache last night weight decreased ...
Name: text, dtype: object
```

[29]:
```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer(r'\w+')
dataset['text'] = dataset['text'].apply(tokenizer.tokenize)
dataset['text'].head()
```

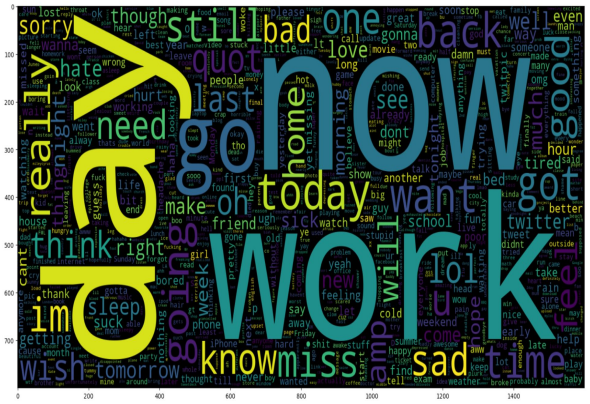
[29…]
```
800000              [love, healthuandpets, u, guys, r, best]
800001    [im, meeting, one, besties, tonight, cant, wai...
800002    [darealsunisakim, thanks, twitter, add, sunisa...
800003    [sick, really, cheap, hurts, much, eat, real, ...
800004                  [lovesbrooklyn, effect, everyone]
Name: text, dtype: object
```

+ Code + Markdown

[ ]:
```
import nltk
st = nltk.PorterStemmer()
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data
dataset['text']= dataset['text'].apply(lambda x: stemming_on_text(x))
dataset['text'].head()
```

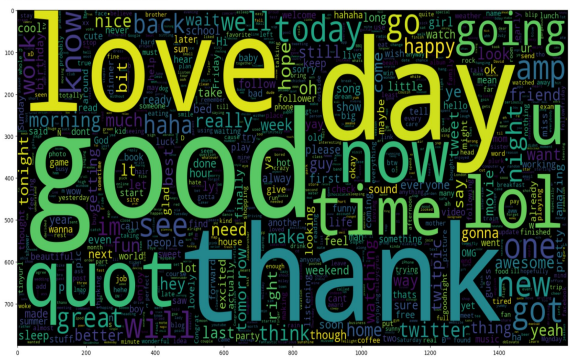
[46]:
```
data_neg = data['text'][:800000]
plt.figure(figsize = (20,20))
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_neg))
plt.imshow(wc)
```

[46…] <matplotlib.image.AxesImage at 0x746c564777d0>



[47]:
```
data_pos = data['text'][800000:]
wc = WordCloud(max_words = 1000 , width = 1600 , height = 800,
               collocations=False).generate(" ".join(data_pos))
plt.figure(figsize = (20,20))
plt.imshow(wc)
```

[47…] <matplotlib.image.AxesImage at 0x746c5639c750>



[54]:
```
# Separating the 95% data for training data and 5% for testing data
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)
```

[55]:
```
vectoriser = TfidfVectorizer(ngram_range=(1,2), max_features=500000)
vectoriser.fit(X_train)
print('No. of feature_words: ', len(vectoriser.get_feature_names()))
```

```
No. of feature_words:  500000
```

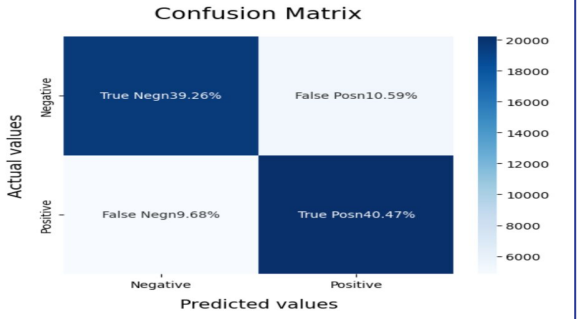
[56]:
```
X_train = vectoriser.transform(X_train)
X_test  = vectoriser.transform(X_test)
```

SVC Model

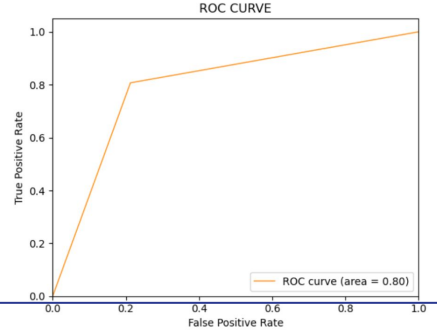
```
SVCmodel = LinearSVC()
SVCmodel.fit(X_train, y_train)
model_Evaluate(SVCmodel)
y_pred2 = SVCmodel.predict(X_test)
```

80%

	precision	recall	f1-score	support
0	0.80	0.79	0.79	24927
1	0.79	0.81	0.80	25073
accuracy			0.80	50000
macro avg	0.80	0.80	0.80	50000
weighted avg	0.80	0.80	0.80	50000




```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred2)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

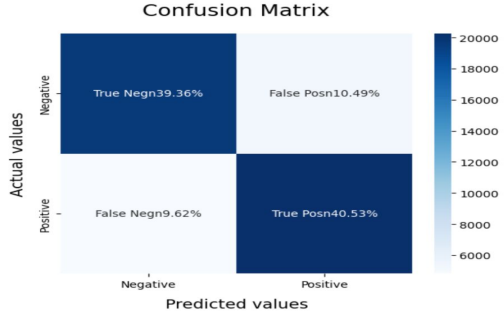


LR Model

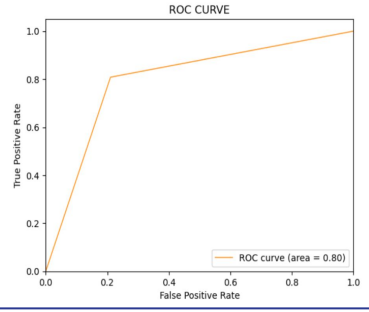
```
LRmodel = LogisticRegression(max_iter = 3000, n_jobs=-1,C=10)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
```

80%

	precision	recall	f1-score	support
0	0.80	0.79	0.80	24927
1	0.79	0.81	0.80	25073
accuracy			0.80	50000
macro avg	0.80	0.80	0.80	50000
weighted avg	0.80	0.80	0.80	50000




```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred3)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

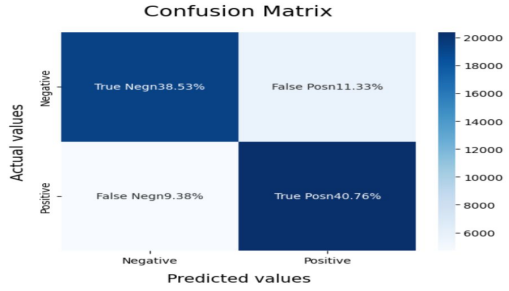


BNB Model

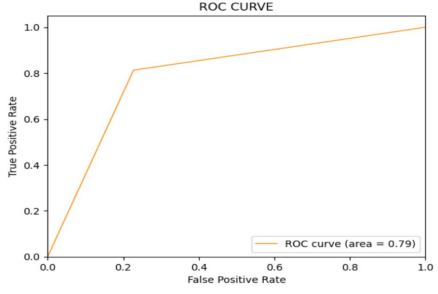
```
BNBmodel = BernoulliNB()
BNBmodel.fit(X_train, y_train)
model_Evaluate(BNBmodel)
y_pred1 = BNBmodel.predict(X_test)
```

79%

	precision	recall	f1-score	support
0	0.80	0.77	0.79	24927
1	0.78	0.81	0.80	25073
accuracy			0.79	50000
macro avg	0.79	0.79	0.79	50000
weighted avg	0.79	0.79	0.79	50000




```
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred1)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

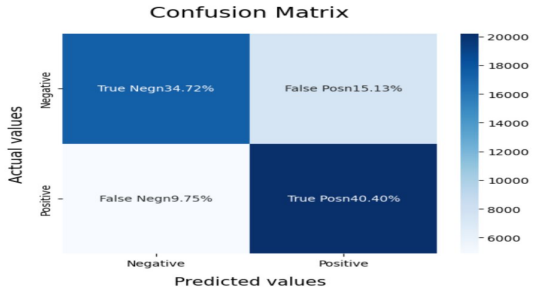


LGBM Model

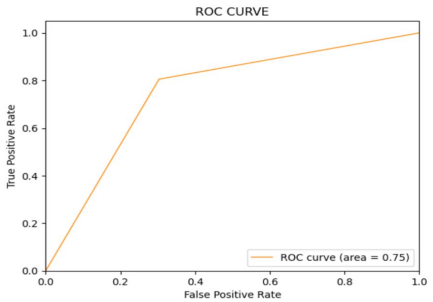
```
[65]:
import lightgbm as lgb
lgbm = lgb.LGBMClassifier(random_state=42)
lgbm.fit(X_train, y_train)
model_Evaluate(lgbm)
y_pred_temp = lgbm.predict(X_test)
```

75%

	precision	recall	f1-score	support
0	0.78	0.70	0.74	24927
1	0.73	0.81	0.76	25073
accuracy			0.75	50000
macro avg	0.75	0.75	0.75	50000
weighted avg	0.75	0.75	0.75	50000




```
[67]:
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred_temp)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

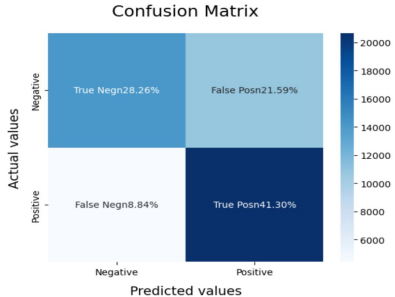


GBC Model

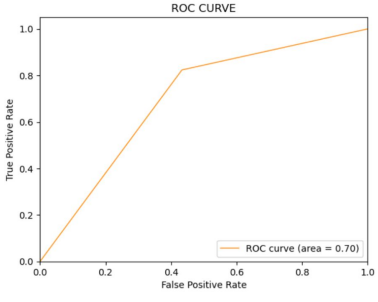
```
[72]:
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gbc.fit(X_train, y_train)
model_Evaluate(gbc)
y_pred6 = gbc.predict(X_test)
```

70%

	precision	recall	f1-score	support
0	0.76	0.57	0.65	24927
1	0.66	0.82	0.73	25073
accuracy			0.70	50000
macro avg	0.71	0.70	0.69	50000
weighted avg	0.71	0.70	0.69	50000




```
[73]:
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred6)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

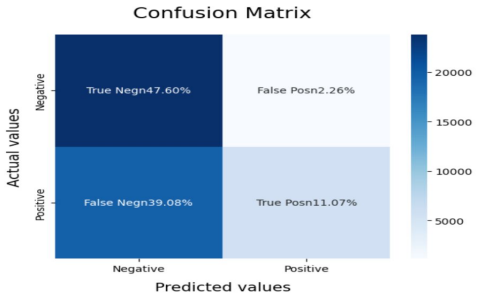


KNN Model

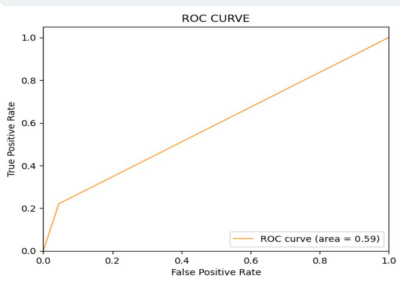
```
[74]:
knn = KNeighborsClassifier(n_neighbors=2)
knn.fit(X_train, y_train)
model_Evaluate(knn)
y_pred7 = knn.predict(X_test)
```

59%

	precision	recall	f1-score	support
0	0.55	0.95	0.70	24927
1	0.83	0.22	0.35	25073
accuracy			0.59	50000
macro avg	0.69	0.59	0.52	50000
weighted avg	0.69	0.59	0.52	50000




```
[75]:
from sklearn.metrics import roc_curve, auc
fpr, tpr, thresholds = roc_curve(y_test, y_pred7)
roc_auc = auc(fpr, tpr)
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVE')
plt.legend(loc="lower right")
plt.show()
```

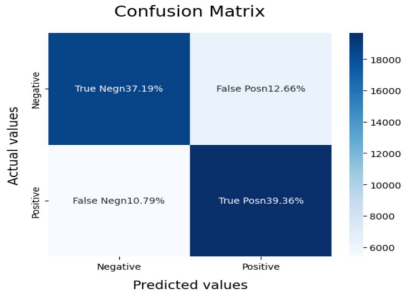


SGD Model

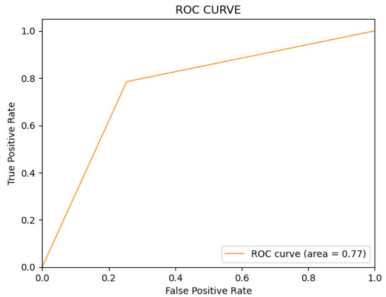
```
[76]: sgd = SGDClassifier(loss='log', alpha=0.0001, max_iter=1000, random_state=42)
      sgd.fit(X_train, y_train)
      model_Evaluate(sgd)
      y_pred8 = sgd.predict(X_test)
```

77%

	precision	recall	f1-score	support
0	0.78	0.75	0.76	24927
1	0.76	0.78	0.77	25073
accuracy			0.77	50000
macro avg	0.77	0.77	0.77	50000
weighted avg	0.77	0.77	0.77	50000




```
[77]: from sklearn.metrics import roc_curve, auc
      fpr, tpr, thresholds = roc_curve(y_test, y_pred8)
      roc_auc = auc(fpr, tpr)
      plt.figure()
      plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
      plt.xlim([0.0, 1.0])
      plt.ylim([0.0, 1.05])
      plt.xlabel('False Positive Rate')
      plt.ylabel('True Positive Rate')
      plt.title('ROC CURVE')
      plt.legend(loc="lower right")
      plt.show()
```

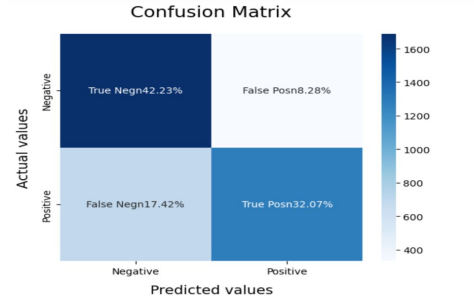


RFC Model

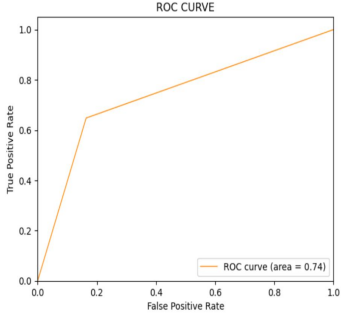
```
rfc = RandomForestClassifier(n_estimators=10, random_state=42)
rfc.fit(X_train, y_train)
model_Evaluate(rfc)
```

74%

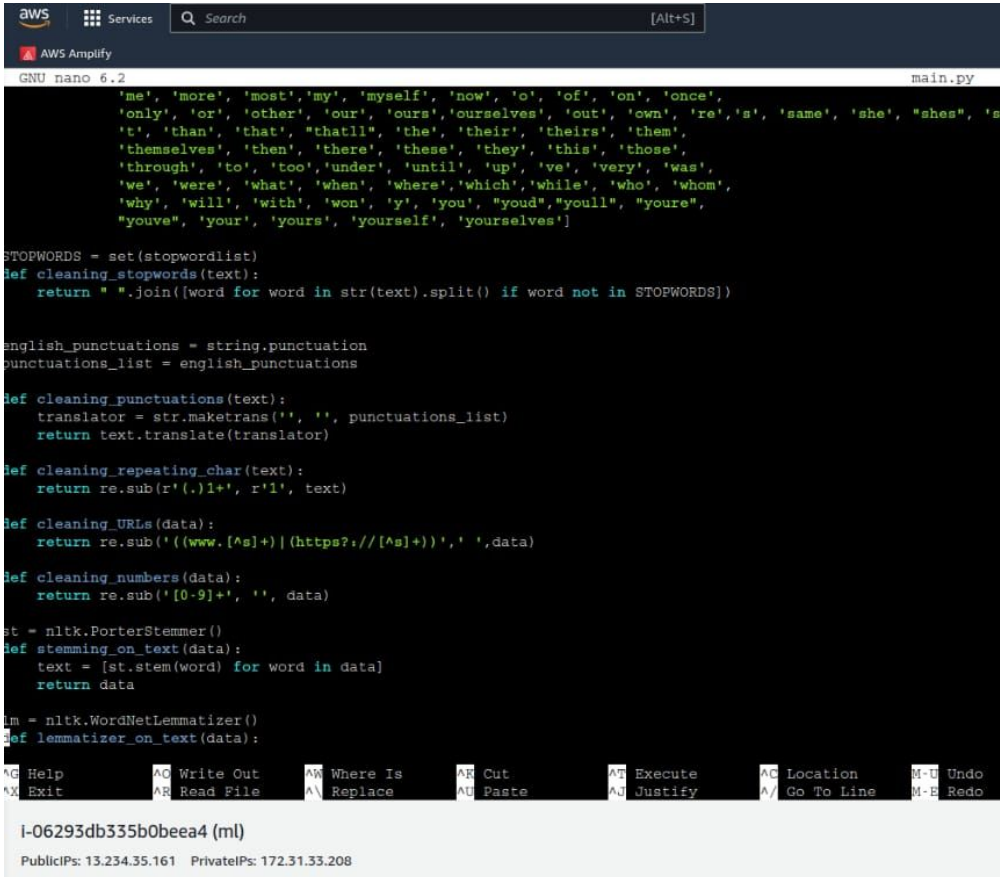
	precision	recall	f1-score	support
0	0.71	0.84	0.77	2020
1	0.79	0.65	0.71	1980
accuracy			0.74	4000
macro avg	0.75	0.74	0.74	4000
weighted avg	0.75	0.74	0.74	4000



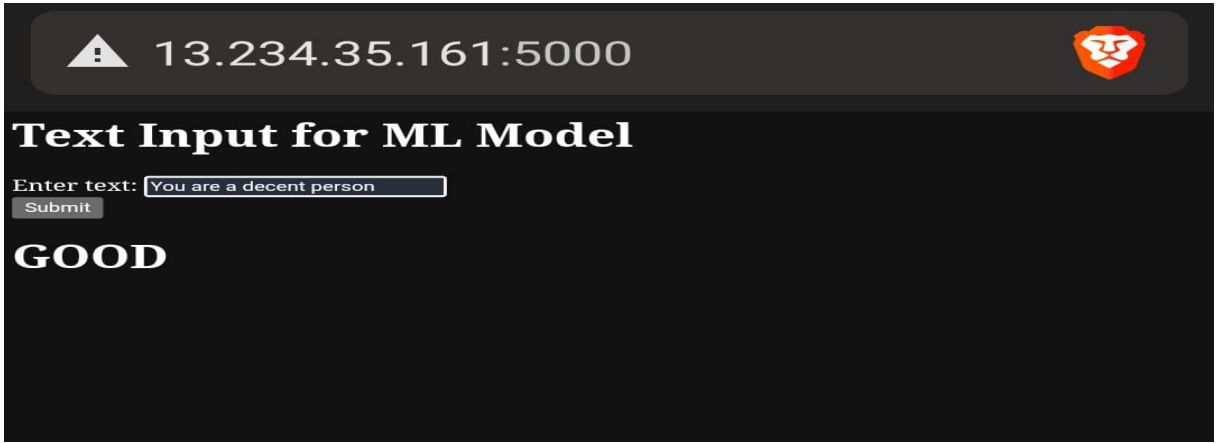

```
[112]: from sklearn.metrics import roc_curve, auc
       fpr, tpr, thresholds = roc_curve(y_test, y_pred5)
       roc_auc = auc(fpr, tpr)
       plt.figure()
       plt.plot(fpr, tpr, color='darkorange', lw=1, label='ROC curve (area = %0.2f)' % roc_auc)
       plt.xlim([0.0, 1.0])
       plt.ylim([0.0, 1.05])
       plt.xlabel('False Positive Rate')
       plt.ylabel('True Positive Rate')
       plt.title('ROC CURVE')
       plt.legend(loc="lower right")
       plt.show()
```

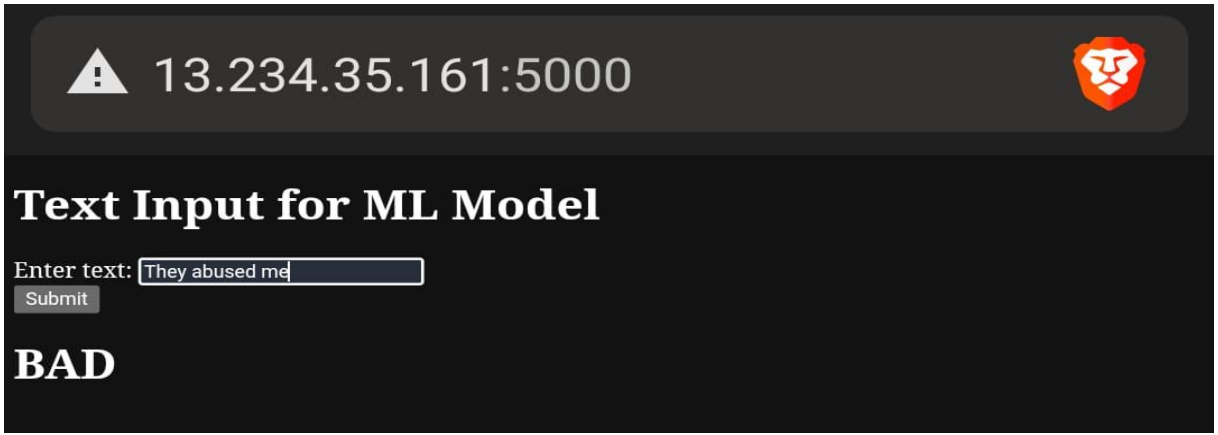


Integration of ML Model on Cloud Computing (AWS):



Output after Integration:





THANK YOU !

Developer Profiles


github/colonel-chirag


github/KSS-10


github/ABS-007


github/divyanksingh-git