JACOBS UNIVERSITY

Advanced Project 1

Sentimental analysis using tweeter data

Prepared by:- Ajay ghimire

Instructor: - Prof. Dr. Adalbert F.X. Wilhelm

Deadline: - August 31, 11:59 pm

Table of content

Introduction

Sentiment analysis is the process of recognizing and categorizing the sentiments represented in a text source. When analysed, tweets are typically beneficial in providing a large volume of sentiment data. These statistics are valuable in determining public opinion on a variety of topics.

To compute the consumer perspective, we must create an Automated Machine Learning Sentiment Analysis Model. It becomes challenging to apply models on them due to the existence of non-useful characters (together referred to as noise) alongside relevant data.

In this paper, we create a machine learning pipeline that uses three classifiers (Logistic Regression, Bernoulli Naive Bayes, and Neural Networks) to analyse the sentiment of tweets from the Sentiment140 dataset.

Overview of dataset

In this research, we attempt to construct a Twitter sentiment analysis model to assist in overcoming the obstacles of recognizing tweet sentiments. The dataset's necessary details are as follows:

The Sentiment140 Dataset is offered, and it consists of 1,600,000 tweets retrieved using the Twitter API. The dataset contains the following columns:

target: positive or negative

ids: The tweet's unique identifier.

date: date of tweet

flag: null if no such query is made

user: person who tweeted

text: It refers to the tweet's text.

Exploratory Data Analysis (EDA)

[24]:
```
# utilities
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.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import confusion_matrix, classification_report
```

[25]:
```
import os
os.getcwd()
```

[25]:
```
'C:\\Users\\ajayg'
```

[26]:
```
# Importing the dataset
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('data.csv', encoding=DATASET_ENCODING, names=DATASET_COLUMNS)
df.sample(5)
```

[26]:

	target	ids	date	flag	user	text
234609	0	1979621524	Sun May 31 03:50:13 PDT 2009	NO_QUERY	dtonk	@tomlambe i suppose we could lol x
1142616	4	1977329476	Sat May 30 20:42:53 PDT 2009	NO_QUERY	HWarrenScott	@Kitty_Gogo Pulling weeds isn't so bad...drink...

```
[28]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1600000 entries, 0 to 1599999
Data columns (total 6 columns):
 #   Column  Non-Null Count    Dtype
---  ------  --------------    -----
 0   target  1600000 non-null  int64
 1   ids     1600000 non-null  int64
 2   date    1600000 non-null  object
 3   flag    1600000 non-null  object
 4   user    1600000 non-null  object
 5   text    1600000 non-null  object
dtypes: int64(2), object(4)
memory usage: 73.2+ MB
```

Fig: - Screenshot of data visualization

```
[37]: # Plotting the distribution for dataset.
      sns.countplot(x='target', data=df)

[37]: <AxesSubplot:xlabel='target', ylabel='count'>
```

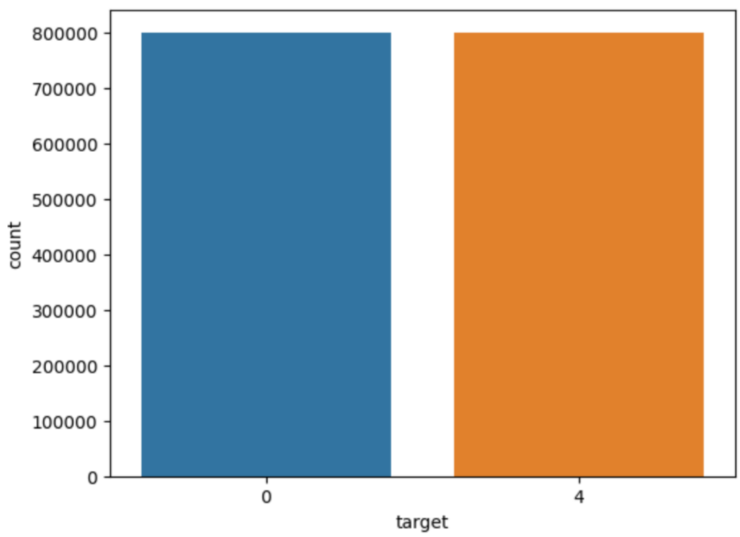

Fig: - Screenshot of data distrubution

Data pre-processing

Before training the model, we did numerous pre-processing steps on the dataset in the problem statement above, which primarily dealt with removing stop words and emojis. For greater generalization, the written document is subsequently transformed to lowercase. Following that, the punctuations were cleaned and eliminated, removing superfluous noise from the dataset. Following that, we deleted the repetitive letters from the words, as well as the URLs, as they are of no significance.

Finally, for better results, we performed Stemming (reducing the words to their derived stems) and Lemmatization (reducing the derived words to their root form known as lemma).

1: Selecting the text and Target column for our further analysis

2.Replacing the values to ease understanding. (Assigning 1 to Positive sentiment 4)

3: Print unique values of target variables

4: Separating positive and negative tweets

6: Combining positive and negative tweets

7: Making statement text in lower case

8: Defining set containing all stopwords in English.

9: Cleaning and removing the above stop words list from the tweet text

10: Cleaning and removing punctuations

11: Cleaning and removing repeating characters

12: Cleaning and removing URL's

13: Cleaning and removing Numeric numbers

14: Getting tokenization of tweet text

15: Applying Stemming

16: Applying Lemmatizer

17: Separating input feature and label

18: Plot a cloud of words for negative tweets

19: Plot a cloud of words for positive tweets

```
[64]: #Data Preprocessing
#Selecting the text and Target column for our further analysis
data=df[['text','target']]
#Replacing the values to ease understanding. (Assigning 1 to Positive sentiment 4)
data['target'] = data['target'].replace(4,1)
#Print unique values of target variables
data['target'].unique()
#Separating positive and negative tweets
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
#taking one fourth data so we can run on our machine easily
data_pos = data_pos.iloc[:int(20000)]
data_neg = data_neg.iloc[:int(20000)]
#Combining positive and negative tweets
dataset = pd.concat([data_pos, data_neg])
```

```
[65]: #Making statement text in lower case
dataset['text']=dataset['text'].str.lower()
dataset['text'].tail()
```

```
[65]: 19995    not much time off this weekend, work trip to m...
19996                         one more day of holidays
19997    feeling so down right now .. i hate you damn h...
19998    geez,i hv to read the whole book of personalit...
19999    i threw my sign at donnie and he bent over to ...
Name: text, dtype: object
```

```
[66]: #Defining set containing all stopwords in English.
#Cleaning and removing the above stop words list from the tweet text
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',
```

```
            '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']
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()
```

```
[66]: 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
```

```
[67]: #Cleaning and removing punctuations
      #Cleaning and removing repeating characters
      #Cleaning and removing URL's

      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()
      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()
      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()

[67]: 19995    not much time off weekend work trip malmï¿½ fr...
```

Fig:- Screenshot of data pre-processing

```
Untitled2.ipynb
Code                                                     Python 3 (ipykernel)
          return re.sub(r'(.)1+', r'1', text)
      dataset['text'] = dataset['text'].apply(lambda x: cleaning_repeating_char(x))
      dataset['text'].tail()
      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()

[67]: 19995    not much time off weekend work trip malmï¿½ fr...
      19996                                     one day holidays
      19997                      feeling right  hate damn humprey
      19998    geezi hv read whole book personality types emb...
      19999    threw sign donnie bent over get but thingee ma...
      Name: text, dtype: object

[68]: #Cleaning and removing Numeric numbers
      #Getting tokenization of tweet text
      #Applying Stemming
      #Applying Lemmatizer
      #Separating input feature and label
      def cleaning_numbers(data):
          return re.sub('[0-9]+', '', data)
      dataset['text'] = dataset['text'].apply(lambda x: cleaning_numbers(x))
      dataset['text'].tail()
      from nltk.tokenize import RegexpTokenizer
      tokenizer = RegexpTokenizer(r'w+')
      dataset['text'] = dataset['text'].apply(tokenizer.tokenize)
      dataset['text'].head()
      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()
      lm = nltk.WordNetLemmatizer()
      def lemmatizer_on_text(data):
          text = [lm.lemmatize(word) for word in data]
          return data
      dataset['text'] = dataset['text'].apply(lambda x: lemmatizer_on_text(x))
      dataset['text'].head()
      X=data.text
      y=data.target
```

Fig:- Screenshot of data pre-processing

Plotting negative word cloud and positive

```
[70]: #Plot a cloud of words for negative tweets
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)
```

```
[70]: <matplotlib.image.AxesImage at 0x1b3280f4850>
```

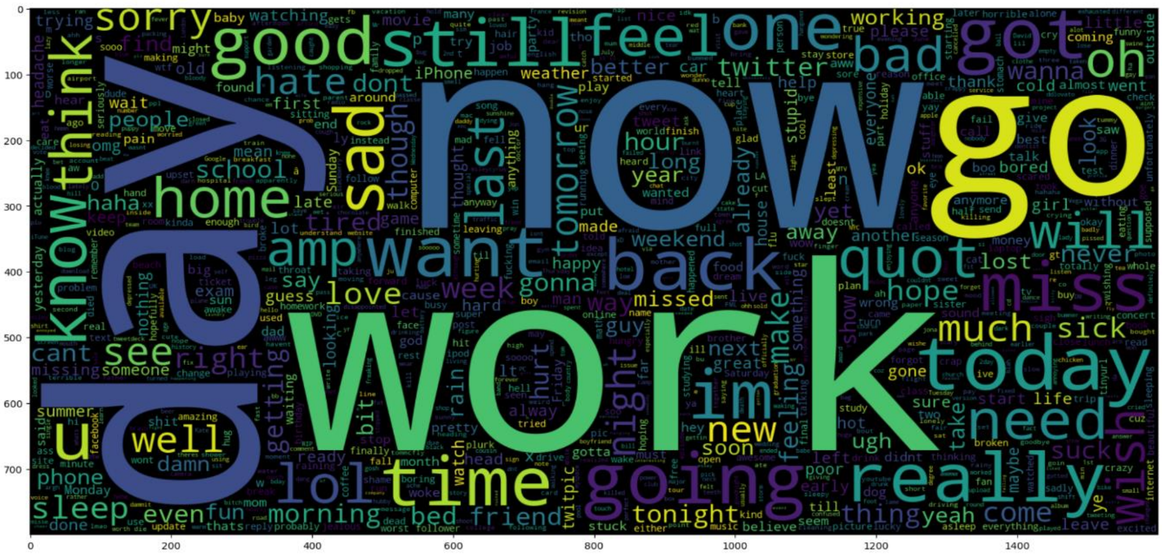

Fig:- Screenshot of negative cloud word

```
[71]: #Plot a cloud of words for positive tweets
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)
```

```
[71]: <matplotlib.image.AxesImage at 0x1b3280f7610>
```

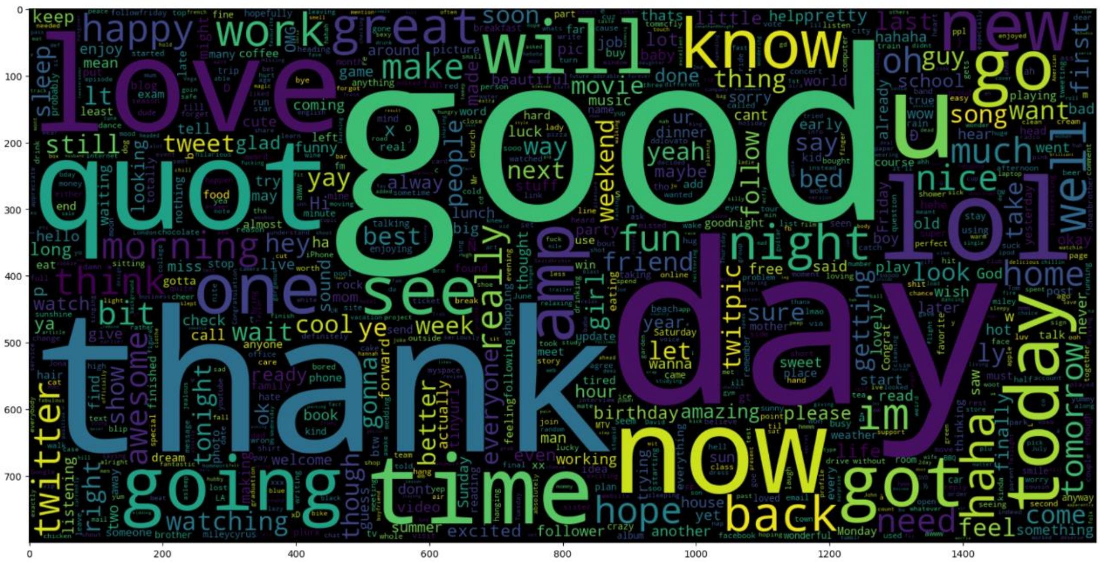

Fig:- Screenshot of positive cloud word

Splitting our data into Train and Test Subset and transforming Dataset using TF-IDF Vectorizer

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

Fig:- Screenshot of splitting the data set into train and test data set

- Naive Bayes Bernoulli
- SVM (Support Vector Machine)
- Regression Logistic

The reason behind selecting these models is that we want to try all of the classifiers on the dataset, from simple to complicated models, and then see which one performs the best.

Naïve Bayes Bernoulli

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

	precision	recall	f1-score	support
0	0.89	0.90	0.90	40097
1	0.67	0.66	0.66	12332
accuracy			0.84	52429
macro avg	0.78	0.78	0.78	52429
weighted avg	0.84	0.84	0.84	52429

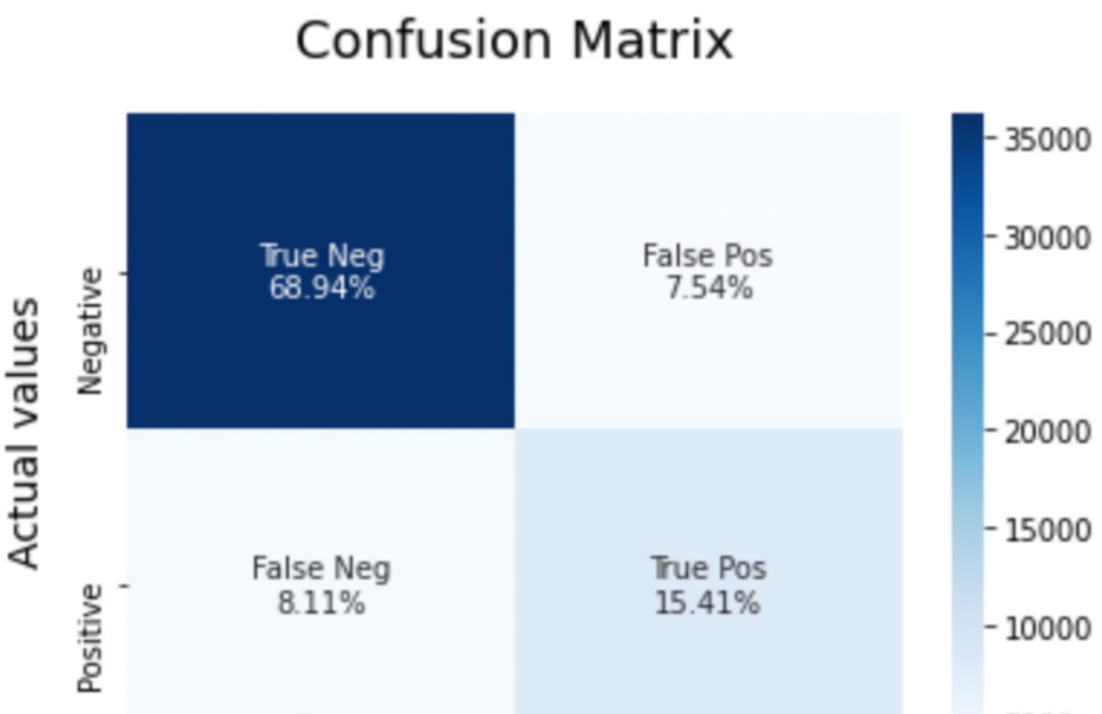

Fig:- Screen of model 1st

SVM (Support Vector Machine)

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

Output:

	precision	recall	f1-score	support
0	0.89	0.93	0.91	40097
1	0.74	0.63	0.68	12332
accuracy			0.86	52429
macro avg	0.81	0.78	0.80	52429
weighted avg	0.86	0.86	0.86	52429

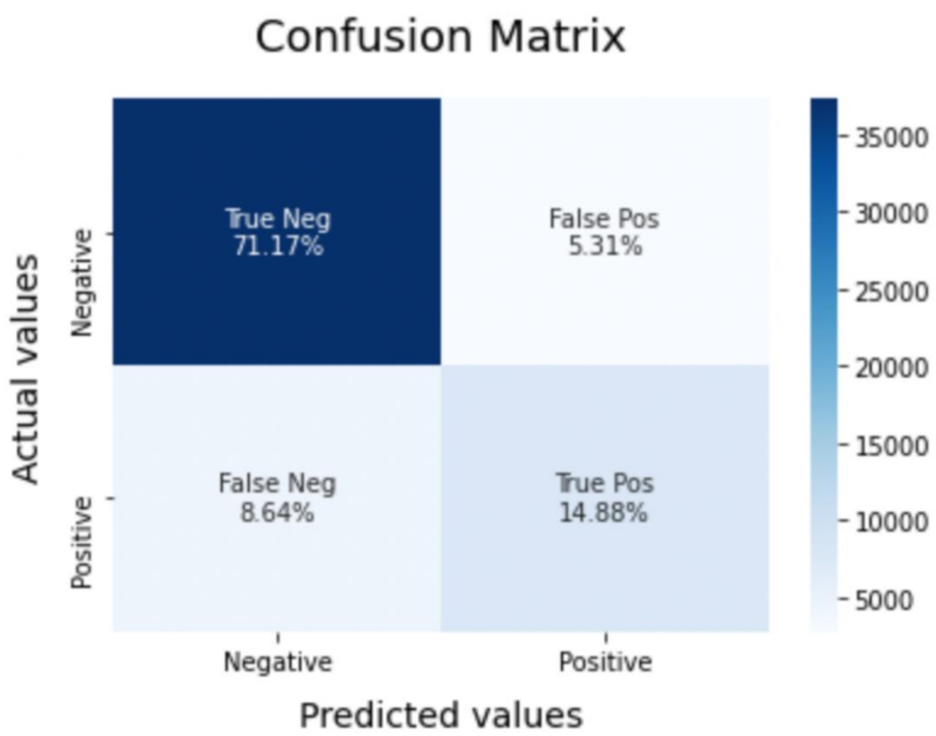

Fig:- Screenshot of Model 2

Logistic Regression

```
LRmodel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
LRmodel.fit(X_train, y_train)
model_Evaluate(LRmodel)
y_pred3 = LRmodel.predict(X_test)
```

Output:

	precision	recall	f1-score	support
0	0.89	0.95	0.92	40097
1	0.78	0.61	0.69	12332
accuracy			0.87	52429
macro avg	0.83	0.78	0.80	52429
weighted avg	0.86	0.87	0.86	52429

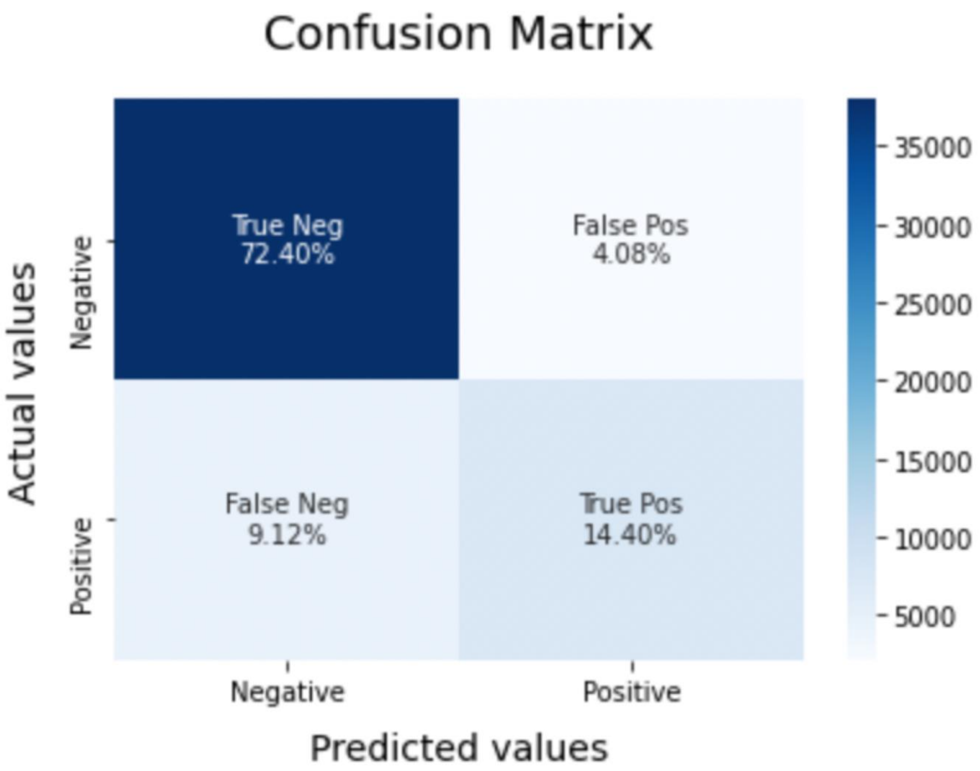

Fig:- Screenshot of Model 3

Conclusion

We can deduce the following details after examining all of the models:

Accuracy: In terms of model accuracy, Logistic Regression outperforms SVM, which outperforms Bernoulli Naive Bayes.

F1-score: The F1 scores for classes 0 and 1 are as follows:

(a) Bernoulli Naive Bayes (accuracy = 0.90) SVM (accuracy = 0.91) Logistic Regression (accuracy = 0.92)

(b) For class 1, the following methods were used: Bernoulli Naive Bayes (accuracy = 0.66) SVM (accuracy = 0.68) Logistic Regression (accuracy = 0.69)

As a result, we find that Logistic Regression is the best model for the aforementioned dataset.

In our problem statement, Logistic Regression follows the idea of Occam's Razor, which states that if the data has no assumptions, then the simplest model works the best. Because our dataset has no assumptions and Logistic Regression is a basic model, the notion applies to the aforementioned dataset.

Reference

1. T. Vijay, P. Karmakar, A. Chawla, and B. Dhanka, "Sentiment analysis on COVID-19 twitter data," in *Proceedings of the 5th IEEE International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, Jaipur, India, December 2020.View at: Publisher Site | Google Scholar

2. S. Das, A. K. Chakraborty, and A. Kumar Kolya, "Sentiment analysis of covid-19 tweets using evolutionary classification-based LSTM model," *Advances in Intelligent Systems and Computing*, Singapore, 2021.View at: Google Scholar

3. S. Bhatia, K. Chakraborty, S. Bhattacharyya, P. Jan, R. Bag, and A. E. Hassanien, "Sentiment Analysis of COVID-19 Tweets by Deep Learning Classifiers—A Study to Show How Popularity Is Affecting Accuracy in Social media," *Applied Soft Computing*, vol. 97, Article ID 106754, 2020.View at: Google Scholar

4. K. Amit, M. Singh, and S. Pandey, "Sentiment Analysis on the Impact of Coronavirus in Social Life Using the BERT Model," *Springer journal of Social Network Analysis and Mining*, vol. 11, no. 1, 2021.View at: Publisher Site | Google Scholar

5. https://www.kaggle.com/datasets/kazanova/sentiment140/code?resource=download
6. https://github.com/cjyanyi/CS542-tweets-sentiment-analysis
7. https://streamhacker.com/2010/05/10/text-classification-sentiment-analysis-naive-bayes-classifier/
8. https://www.analyticsvidhya.com/blog/2021/12/sentiment-analysis-on-tweets-with-lstm-for-beginners/