SENTIMENT ANALYSIS OF TWEETS

Submitted By GroupX:

Chandan Goel

Gopikrishnan Srinivasan

Hitesh Yadav

Sanjeev Singh

Shubhendu Bhaskar

Table of Contents

SENTIMENT ANALYSIS OF TWEETS

Preface

With quick and easy access to the internet through cell phones, people are not hesitant anymore to express themselves over social media on a global scale. Users are expressing themselves be it the presidential election or review of a single dollar product.

This is one of the major reasons that Twitter has become the gargantuan barn of opinions and emotions therefore, product owners, political parties, government agencies, etc. track the viewpoints of the intended audiences via this social media giant. Consequently, there comes a need that requires examination and understanding of the sentiment associated with the tweets that depict the emotional inclination of the individuals. This paves the path to sentiment analysis.

Why Sentiment Analysis of Tweets?

We are currently witnessing the technological boom in all sphere humans have stepped on. In the last two decades, the tech around us has not only shown the advantages of its power but also has drawn several disadvantages towards humankind. For instance, the intent of Twitter creation was to connect its users and allow them to share their thoughts with their followers and others through the use of hashtags (cited) but not long ago, we discovered the dark side of this application. A few of such impacts are:

- Mental Health: Depression, Anxiety, Insomnia, etc.
- Cyberbullying
- Fear of Missing Out (FOMO)
- Unrealistic Expectations
- Negative Body Image
- General Addiction
- Political Polarization

Therefore, with both positive and negative impacts, the Twitter application aligns very well with our objective of classifying a piece of text into positive or negative emotions. This kind of analysis can be further extended by individuals to understand and weigh opinions and make judgments accordingly.

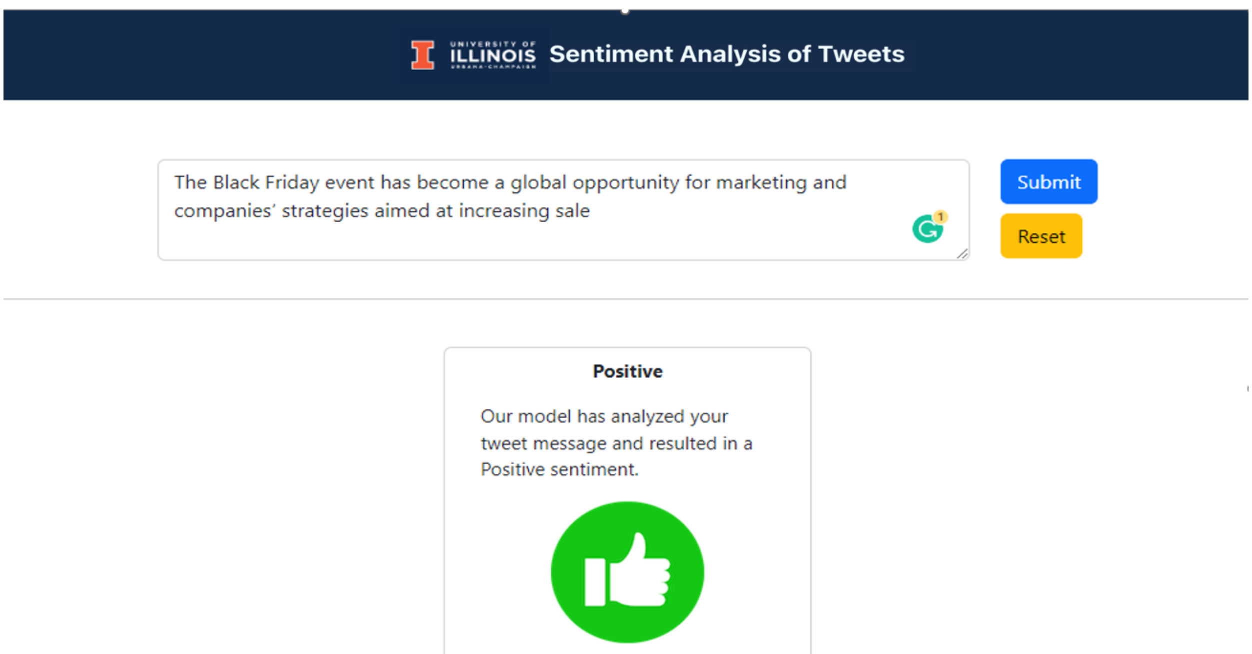
1. Overview

The application (tool) developed allows user to enter a text (tweet) on the screen and computationally determines the sentiment of the tweet for the intended audience.

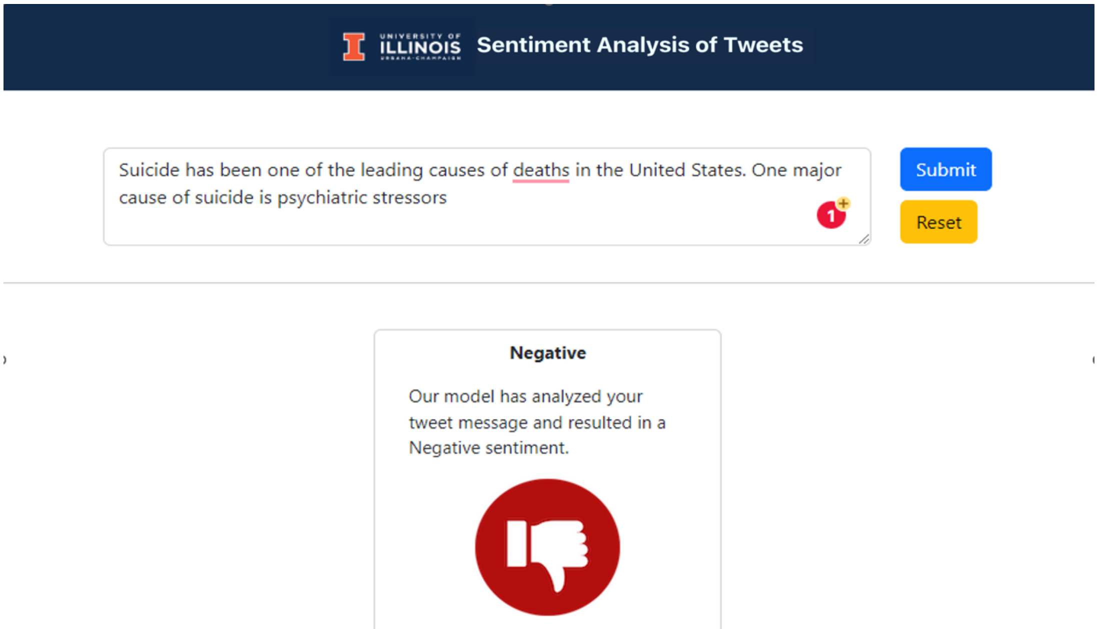
This tool can be used by different stake holders in overviewing the sentiments for a specific objective and understanding the generic opinion trend. For instance, a government may extend this application and integrate tweets (or text from any other application) to assess the public opinion about the policy they want to bring to the congress for its citizen's welfare.

Example of the application output:

Positive tweet sentiment analysis:



Negative tweet sentiment analysis:



2. How to use the software?

The user interface of the application is quite straightforward. The user is required to enter the tweet in the provided text box. Click the "Submit" button to analyze the tweet and produce the corresponding analysis output.

-

a) Software Requirements:

i. Visual Studio Code (IDE)

ii. Python v3.10.6

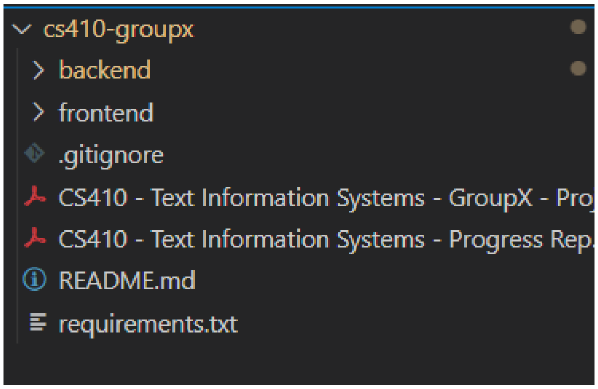
iii. Node.js v19

iv. Npm v8.19

v. Data Source at [Kaggle] (expected format)

b) Code execution:

i. Download the code from the [link]. As shown below root directory has two folders "backend" and "frontend"



ii. "backend" folder contains the code of sentiment analysis engine while "frontend" holds UI code

iii. Install the required dependencies using "*pip install -r requirements.txt"* command. File "requirements.txt" lists the dependencies along with the version.

iv. Go to the directory backend "*cd backend*"

v. Run the modules of Natural Language Toolkit: "*python nltk_modules.py"*

vi. Run the server using "*python server.py"*

vii. Go back one level from the current "backend" directory using "*cd..*" and then change the current directory to "frontend": "*cd frontend"*

viii. Install npm: "*npm install"*

ix. Run the app: *"npm start"*

x. Fire up a browser and hit the url http://localhost:3000

3. How is Software implemented?

As already mentioned in the aforesaid section, Python has been chosen as the coding language in this project. The UI is developed in ReactJS.

Following are the steps for building the twitter sentiment analysis:

a) Importing required Dependencies

Below are the screenshots of the different dependencies used in Twitter Sentiment implementation.

README.md | data_preprocessing.py ×

cs410-groupx > backend > data_preprocessing.py > ...

```python
# utilities
import re
import numpy as np
import pandas as pd
import string

# nltk
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer

```

README.md × | get_sentiments.py M ×

cs410-groupx > backend > get_sentiments.py > ...

```python
import pickle
import pandas as pd
from data_preprocessing import *

```

README.md | get_tweets.py ×

cs410-groupx > backend > get_tweets.py > ...

```python
import configparser
import pandas as pd
import tweepy
from datetime import datetime
```

README.md × *sentiment_analysis.py* ×

cs410-groupx > backend > sentiment_analysis.py > ...

```python
# utilities
import re
import numpy as np
import pandas as pd
import string

# nltk
import nltk
from nltk.stem import WordNetLemmatizer
from nltk.corpus import stopwords
from nltk.tokenize import RegexpTokenizer

# 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

# Pickle
import pickle
```

README.md × *server.py* ×

cs410-groupx > backend > server.py > ...

```python
from flask import Flask, jsonify, request
from flask_cors import CORS
from get_sentiments import *

```

README.md × train_and_create_model.py ×

cs410-groupx > backend > train_and_create_model.py > ...

```python
# import data cleaning
from data_preprocessing import *

# sklearn
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer

# pickel
import pickle
```

b) Reading and Loading the Kaggle Dataset

The source data can be found at [kaggle]:

This link describes context and content of data points, such as the polarity of the target (0 = negative, 2 = neutral, 4 = positive), date of the tweet, user, text of tweet etc. In our project we will be focusing only on *negative* and positive *polarities*.

README.md train_and_create_model.py ×

cs410-groupx > backend > train_and_create_model.py > ...

```python
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('data/training.1600000.processed.noemoticon.csv', encoding=DATASET_ENCODING, names=DATASET_COLUMNS)

```

c) Data Curation and Pre-processing

In this analysis the only columns chosen are "target" and "text"; the other columns like user, ids, date etc. are irrelevant in sentiment prediction therefore, these columns in the datasets are ignored.

In order to make things simple we replaced the target code of "positive" tweets from "4" to "1". Negative tweets are still represented by 0.

```
# Select required column fro mthe dataset
data = df[['text', 'target']]
data['target'] = data['target'].replace(4,1)
```

Post this replacement the data is cleaned before training the model.

To ensure proper data processing below steps are followed:

i. Cleaning the urls – used regular expression
ii. Removing the punctuations – used translation table and str.translate
iii. Remove repeating characters – used regular expression
iv. Converted entire text in lower case
v. Cleaned stopwords - used natural language toolkit (nltk)
vi. Remove numbers – used regular expression
vii. Tokenize each word in the remaining text – used RegexpTokenizer
viii. Stemming – used natural language toolkit (nltk.WordNetLemmatizer)
ix. Lemmatization - used natural language toolkit (nltk.WordNetLemmatize)

cs410-groupx > backend > data_preprocessing.py > clean_data

```python
STOPWORDS = set(stopwords.words('english'))
st = nltk.PorterStemmer()
lm = nltk.WordNetLemmatizer()

# Remove URLs
def cleaning_URLs(data):
    return re.sub('((www.[^s]+)|(https?://[^s]+))',' ',data)

# Remove punctuation
# english_punctuations = string.punctuation
# punctuations_list = english_punctuations

def cleaning_punctuations(text):
    translator = str.maketrans('', '', string.punctuation)
    return text.translate(translator)

# Remove repeating characters
def cleaning_repeating_char(text):
    return re.sub(r'(.)1+', r'1', text)

# Remove stop words
def cleaning_stopwords(text):
    return " ".join([word for word in str(text).split() if word not in STOPWORDS])

# Remove numbers
def cleaning_numbers(data):
    return re.sub('[0-9]+', '', data)

# Stemming
def stemming_on_text(data):
    text = [st.stem(word) for word in data]
    return data

# Lemmatize
def lemmatizer_on_text(data):
    text = [lm.lemmatize(word) for word in data]
    return data
```

d) Extract tweets (features) and the corresponding targets (labels)

From the cleaned dataset, fetch the tweet and the corresponding polarity (target)

```python
# Split dataset features and labels
X = dataset['text']
y = dataset['target']
```

e) Split dataset into train and test datasets

Next, we split the clean data into training and test set.

The data is split using the test size of 0.05 and a random state. X represents the “text” and Y represents the target (positive or negative). Therefore, we see X_Train, X_Test, Y_Train and Y_Test.

Test and train datasets are assigned rows randomly to ensure they are a random sample of the original dataset and represent observation samples. This is achieved by using the random state variable which is an integer value.

For this project, we have split the dataset so that 95% is used to train the model and 5% is used to evaluate the model. The split percentage was chosen arbitrarily.

```
# This is to check the model
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)
```

f) Create, train, and transform the Model

Utilize Term Frequency - Inverse Document Frequency Vectorizer(TfidfVectorizer) to remold the dataset. This will not only consider the word occurrences in a single tweet but in the entire range of tweets. This method will assign lower weights to common words while giving higher weights to rare words but appearing multiple times across tweets. Used sklearn.feature_extraction.text.TfidfVectorizer to transform the X_Train and X_Test (“text”) into Vectorizer.

vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=500000)

(1, 2) means unigrams and bigrams values that are used. Max_features allow to build vocabulary of top 500000 features ordered by term frequency across all tweets.

The vectorizer transforms text into vectors so that they can be used as input to the estimator.

Post pre-modelling, we are using Logistic Regression algorithm on the datasets to build the model that helps to classify the tweets as positive or negative. Logistic regression will predict the binary outcome as 0 – negative tweet and 1-positive tweet.

This is done with Inverse of regularization strength of 2, max iterations of 1000 and then fit the vectorized X_Train against Y_Train. Using regularization strength of 2, we are applying penalty to increase parameter value magnitudes to reduce overfitting. The tf idf fit helps in normalizing the counts of occurrence of words across tweets. Both the outputs of the vectorizer and the train model is dumped into individual /pickle file. Here, pickle files allow us to keep track of tweets serialized so later references will not be serialized again thus achieving faster execution time.

README.md | train_and_create_model.py ×

cs410-groupx › backend › train_and_create_model.py › ...

```
# This is to check the model
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)

vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=500000)

vectorizer.fit(X_train, y_train)
X_train_vectorized = vectorizer.transform(X_train)
X_test_vectorized  = vectorizer.transform(X_test)

lrModel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
lrModel.fit(X_train_vectorized, y_train)

vector_filename = 'textVetorizer.pickle'
model_filename = 'lrModel_trained.pickle'

pickle.dump(vectorizer, open(vector_filename, 'wb'))
pickle.dump(lrModel, open(model_filename, 'wb'))
```

g) Model Evaluation

Once model training is completed, the next step is to evaluate the performance of the model. A classification report is built for evaluation measures.

```python
def model_Evaluate(model):
# Predict values for Test dataset
    y_pred = model.predict(X_test_vectorized)
    # Print the evaluation metrics for the dataset.
    print(classification_report(y_test, y_pred))

# bnbmodel = BernoulliNB()
# bnbmodel.fit(X_train_vectorized, y_train)
# model_Evaluate(bnbmodel)
# y_pred1 = bnbmodel.predict(X_test_vectorized)

lrModel = LogisticRegression(C = 2, max_iter = 1000, n_jobs=-1)
lrModel.fit(X_train_vectorized, y_train)

model_filename = 'lrModel_trained.sav'
pickle.dump(lrModel, open(model_filename, 'wb'))

sampleTweet = ["I like machine learning."]
inputData = pd.DataFrame(sampleTweet, columns=['text'])
```

The trained model is consequently used for making prediction of the input tweet.

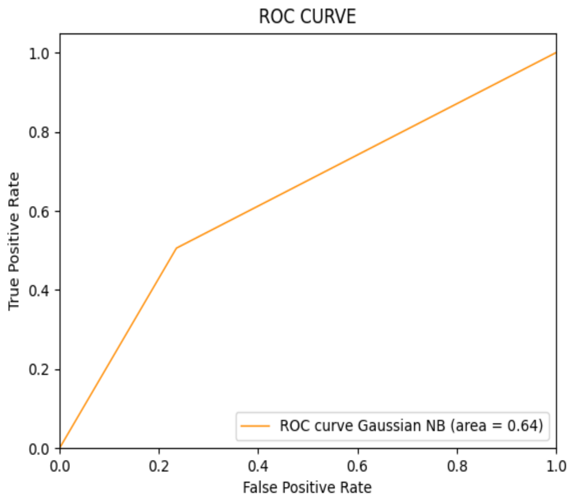
```python
prediction = model.predict(predictForVectorized)

print("Prediction for tweet '{}': {}".format(tweet, prediction))

return prediction
```

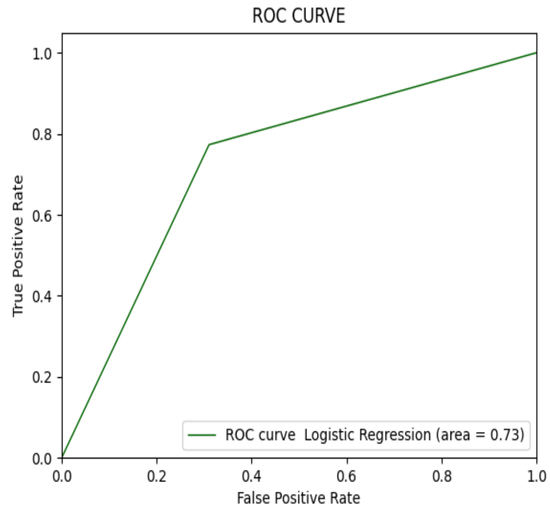
4. Model Performance Evaluation

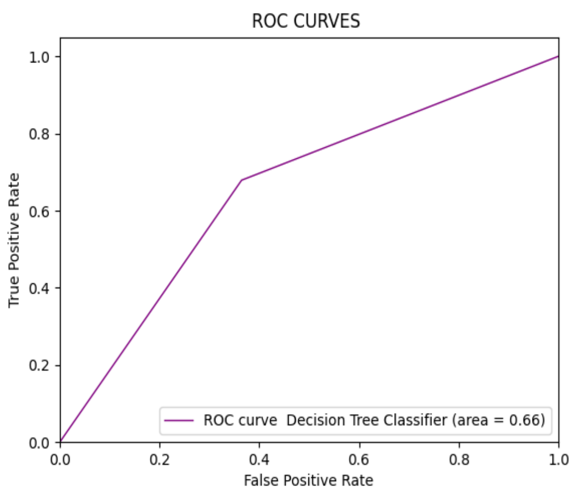
To compare different models based on their performance, ROC curves for 5 different models were generated. Based on the ROC plots shown below, it was confirmed that for Twitter Sentiment Analysis, Logistic Regression model is the best suited model as compared to the other models.

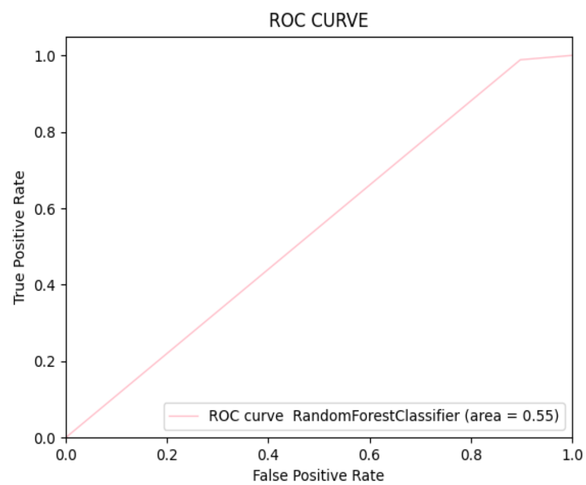
Following were the models chosen for comparison:

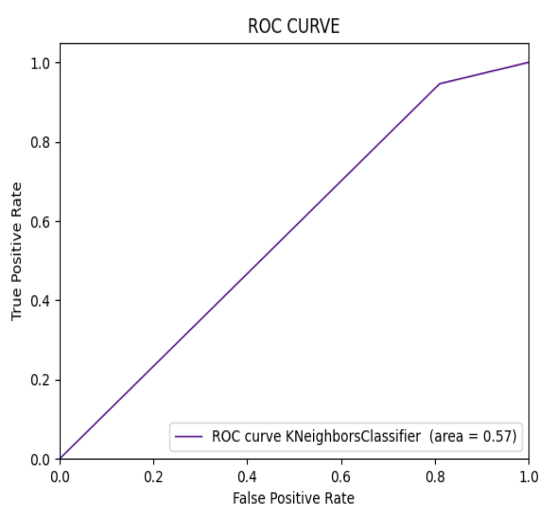
- Gaussian Naïve Bayes
- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- K-Neighbors Classifier


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve Gaussian NB (area = 0.64)


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve Logistic Regression (area = 0.73)

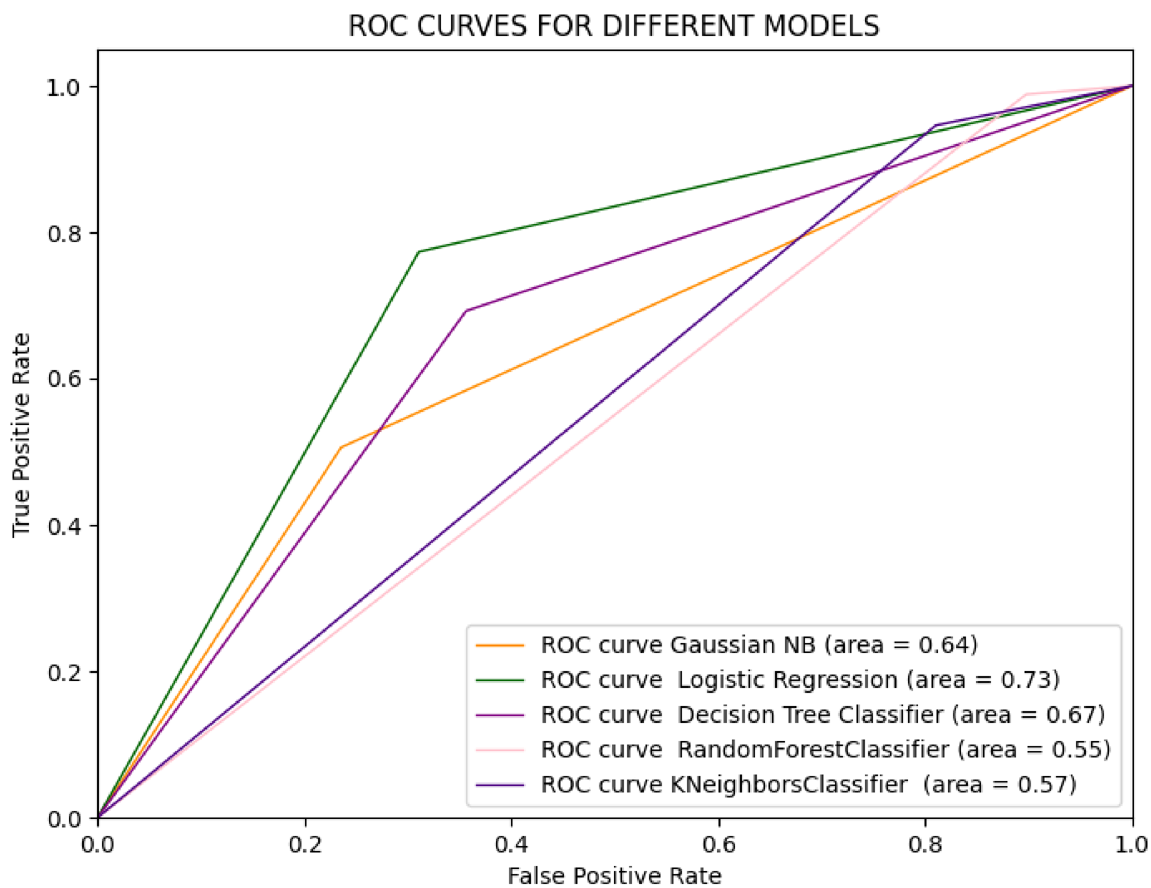

ROC CURVES
True Positive Rate
False Positive Rate
ROC curve Decision Tree Classifier (area = 0.66)


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve RandomForestClassifier (area = 0.55)


ROC CURVE
True Positive Rate
False Positive Rate
ROC curve KNeighborsClassifier (area = 0.57)

The order of ROC AUC Score for the models:

Logistic Regression (0.73) > Decision Tree Classifier (0.67) > Gaussian NB (0.64) > K-Neighbors Classifier (0.57) > Random Forest Classifier (0.55)



For this project, the Logistic Regression model has a chance of performing sentiment analysis with 73% chance.

Code Implementation: ModelComparison.py file

ModelComparison.py U

cs410-groupx > backend > ModelComparison.py > ...

```python
import pandas
import matplotlib.pyplot as plt
from data_preprocessing import *
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import roc_curve, auc
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier

# we are only using a single dataset at this link so we can hardcode some stuff for training.
DATASET_COLUMNS=['target','ids','date','flag','user','text']
DATASET_ENCODING = "ISO-8859-1"
df = pd.read_csv('C:/Users/User/Documents/tis/FinalProject/cs410-groupx/backend/training.1600000.processed.noemoticon.csv', encoding=

# Select required column fro mthe dataset
data = df[['text', 'target']]
data['target'] = data['target'].replace(4,1)


# Reduce dataset
data_pos = data[data['target'] == 1]
data_neg = data[data['target'] == 0]
data_pos = data_pos.iloc[:int(21000)]
data_neg = data_neg.iloc[:int(21000)]
dataset = pd.concat([data_pos, data_neg])
```

```python
# The same function should be called for prediction when taking data from UI
dataset = clean_data(dataset)
# print(dataset['text'].head())

# Split dataset features and labels
X = dataset['text']
y = dataset['target']

#models
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.05, random_state =26105111)
vectorizer = TfidfVectorizer(ngram_range=(1,2), max_features=10000)

vectorizer.fit(X_train, y_train)
X_train_vectorized = vectorizer.transform(X_train)
X_test_vectorized  = vectorizer.transform(X_test)

#GaussianNB
GNB = GaussianNB()
GNB.fit(X_train_vectorized.toarray(),y_train)
y_pred1 =GNB.predict(X_test_vectorized.toarray())

#Logistic Regression
LR = LogisticRegression()
LR.fit(X_train_vectorized.toarray(), y_train)

y_pred2 = LR.predict(X_test_vectorized.toarray())

# Decision Tree Classifier
DTC = DecisionTreeClassifier()
DTC.fit(X_train_vectorized.toarray(), y_train)
```

```python
y_pred3 = DTC.predict(X_test_vectorized.toarray())

#Random Forest Classifier
RFC = RandomForestClassifier(random_state=42, n_jobs=-1, max_depth=5, n_estimators=100, oob_score=True)
RFC.fit(X_train_vectorized.toarray(), y_train)
y_pred4 = RFC.predict(X_test_vectorized.toarray())

#K Neighbors Ckassifier
KN = KNeighborsClassifier(n_neighbors =3)
KN.fit(X_train_vectorized.toarray(), y_train)
y_pred5 = KN.predict(X_test_vectorized)

fp1, tp1, thresholds1 = roc_curve(y_test, y_pred1)
roc_auc1= auc(fp1, tp1)
fp2, tp2, thresholds2 = roc_curve(y_test, y_pred2)
roc_auc2= auc(fp2, tp2)
fp3, tp3, thresholds3 = roc_curve(y_test, y_pred3)
roc_auc3= auc(fp3, tp3)
fp4, tp4, thresholds4 = roc_curve(y_test, y_pred4)
roc_auc4= auc(fp4, tp4)
fp5, tp5, thresholds5 = roc_curve(y_test, y_pred5)
roc_auc5= auc(fp5, tp5)

plt.figure()
plt.plot(fp1, tp1, color='darkorange', lw=1, label='ROC curve Gaussian NB (area = %0.2f)' % roc_auc1)
plt.plot(fp2, tp2, color='darkgreen', lw=1, label='ROC curve  Logistic Regression (area = %0.2f)' % roc_auc2)
plt.plot(fp3, tp3, color='purple', lw=1, label='ROC curve  Decision Tree Classifier (area = %0.2f)' % roc_auc3)
plt.plot(fp4, tp4, color='pink', lw=1, label='ROC curve  RandomForestClassifier (area = %0.2f)' % roc_auc4)
plt.plot(fp5, tp5, color='indigo', lw=1, label='ROC curve KNeighborsClassifier  (area = %0.2f)' % roc_auc5)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
```

```python
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC CURVES FOR DIFFERENT MODELS')
plt.legend(loc="lower right")
plt.show()
```

5. Description of Team Contribution

Team Member	Project Contribution
Chandan Goel	Project Proposal and Progress Report Documentation
Gopikrishnan Srinivasan	Initial data analysis, User Interface creation using ReactJS and Bootstrap, Integration with the backend ML model using Flask (Python)
Hitesh Yadav	Data curation, machine learning modeling and prediction code
Sanjeev Singh	Presentation PowerPoint
Shubhendu Bhaskar	Initial Data Analysis, UI research and front end POC built in Dash, ROC-AUC models comparison programming and Final Project Report

6. Related Work

Different related work was referenced during the project.

Projects from our UIUC university:

https://mediaspace.illinois.edu/media/t/1_b5o4qhuo/112201961

NLP user case for beginners:

https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/

7. References

https://scikit-learn.org/stable/model_persistence.html

https://www.skillfinder.com.au/course/what-is-the-main-purpose-of-twitter