AML 2203 - Advanced Python AI and ML Tools
Final Project Report
Twitter Data Analysis

GROUP - 7

Submitted to: Prof. Manasa Ram

Submitted by : Group 5

- ISHIKA SUKHIJA (C0895302)
- NIPUN GULATI (C0896042)
- REWANT SHARMA (C0894265)
- SAURABH GANGWAR (C0894380)
- ROHIT KUMAR (C0895100)

Table of Contents

INTRODUCTION

Sentiment analysis is a type of contextual mining that helps businesses monitor online conversations to understand the social sentiment surrounding their brand, product, or service by identifying and extracting subjective information from source material. Nevertheless, basic sentiment analysis and count-based metrics are typically the only ones used to analyze social media streams. This is like only grazing the tip of the iceberg and losing out on those high-value discoveries that are lying in wait. What then ought to a brand do to seize that low-hanging fruit?

Recent developments in deep learning have significantly enhanced algorithms' capacity for text analysis. Advanced artificial intelligence algorithms used creatively can be a useful tool for conducting in-depth research. We think it's critically to categorize incoming consumer discussions regarding a brand according to the lines that follow:

1. Important features of a brand's goods and services that appeal to consumers.
2. The underlying intentions and responses of users regarding those elements.

Why is sentiment analysis relevant? :
Opinion mining, also known as sentiment analysis, is an essential business intelligence method that helps companies improve their products and services. The following is a list of a few sentiment analysis benefits.

Offer a dispassionate evaluation. Employing sentiment analysis tools powered by artificial intelligence (AI) can help organizations avoid the personal prejudice that comes with using human reviewers. Consequently, companies that evaluate consumer feedback obtain reliable and unbiased information.

<u>Take the following sentence, for instance:</u>
I'm amazed by the speed of the processor but disappointed that it heats up quickly.

Marketers might dismiss the discouraging part of the review and be positively biased towards the processor's performance. However, accurate sentiment analysis tools sort and classify text to pick up emotions objectively.

Build better products and services:
With the use of accurate and sincere customer feedback, a sentiment analysis system helps companies improve their products and services. Artificial intelligence (AI) systems identify situations or actual objects (referred to as things) that make customers feel bad. Product developers should focus on improving the processor's heat management capability because the text analysis application negatively affects the processor and raises its temperature.

Analyze at scale:
Enterprises continuously extract insights from an abundance of unstructured data, including chatbot

transcripts, emails, surveys, customer relationship management data, and product reviews. Businesses may scale the process of identifying client emotions in textual data at a reasonable cost by utilizing cloud-based sentiment analysis tools.

Real-time results
Businesses must be quick to respond to potential crises or market trends in today's fast-changing landscape. Marketers rely on sentiment analysis software to learn what customers feel about the company's brand, products, and services in real time and take immediate actions based on their findings. They can configure the software to send alerts when negative sentiments are detected for specific keywords.

What are sentiment analysis use cases?
Enterprises employ sentiment analysis to extract insights and formulate feasible strategies across several domains.

Improve customer service:
Customer support representatives can use sentiment analysis techniques to customize their responses based on the conversation's tone. Chatbots powered by artificial intelligence (AI) and equipped with sentiment analysis tools recognize critical problems and route them to support personnel.

Brand monitoring:
Customer support representatives can use sentiment analysis techniques to customize their responses based on the conversation's tone. Chatbots powered by artificial intelligence (AI) and equipped with sentiment analysis tools recognize critical problems and route them to support personnel.

Market research:
A sentiment analysis system helps businesses improve their products by determining what works and what doesn't. Through social media posts, survey responses, and comments on online review sites, marketers can gain deeper insights into certain product attributes. The product engineers are informed of the outcomes, and they modify their invention accordingly.

Track campaign performance:
Marketers utilize sentiment analysis technology to ensure that their campaign receives the intended response. They keep an eye on social media discussions to ensure that the dialogue is generally constructive. If the net sentiment is not as high as anticipated, marketers modify the campaign based on real-time data analytics.

How does sentiment analysis work?

Natural language processing (NLP) technologies are used in sentiment analysis to teach computer programs to comprehend text in a manner akin to that of humans. Usually, the analysis goes through multiple phases before yielding the ultimate outcome.
Preprocessing
Sentiment analysis finds important terms to emphasize the main ideas in the text in the preprocessing phase.

- A sentence is tokenized, or divided into many parts.
- Words are lemmatized to reveal their root form. As an illustration, the root form of am be.
- Stop-word elimination eliminates words from sentences that don't significantly improve it. Words like with, for, at, and of are examples of stop words.

Keyword analysis
The collected keywords are further examined by NLP technologies, which also provide a sentiment score to them. A sentiment analysis system's emotional component is shown by a sentiment score, which is a measurement scale. For analytical reasons, it offers a relative sense of the emotion represented in the text. For instance, when examining client feedback, analysts utilize a scale of 10 for satisfaction and 0 for dissatisfaction.

What are the approaches to sentiment analysis?
<u>There are three main approaches used by sentiment analysis software.</u>

1.Rule-based
The rule-based method uses preset lexicons to identify, categorize, and score particular terms. Lexicons are collections of terms that express the intention, feelings, and attitude of the author. In order to represent the emotional weight of various terms, marketers give sentiment scores to both positive and negative lexicons. The program looks for terms included in the lexicon and totals the sentiment score to decide whether a sentence is favorable, negative, or neutral. To ascertain the total emotional bearing, the final score is compared to the sentiment bounds.

2.Rule-based analysis example
Imagine a system where the positive lexicon has words like joyful, inexpensive, and quick, whereas the negative lexicon contains words like difficult, costly, and impoverished. Marketers use a scale of -1 to -10 for negative word scores and 5 to 10 for positive word scores. To recognize double negatives as positive sentiments, like "not bad," special restrictions are put in place. Marketers determine that a sentiment score between -3 and 3 is classified as mixed, and any score above 3 is considered favorable.

Pros and cons:
Setting up a rule-based sentiment analysis system is simple, but scaling it is challenging. For instance, when you find new keywords for expressing purpose in the text input, you'll need to keep adding to the lexicons. Furthermore, this method might not work well when processing sentences that have cultural influences.

ML:
This method trains computer software to recognize emotional sentiment in text by utilizing machine learning (ML) techniques and sentiment classification algorithms, such as neural networks and deep learning. In order to train a sentiment analysis model to accurately infer sentiment from unknown data, it must first be created and then frequently trained on known data.

Training
Data scientists employ large-scale example datasets for sentiment analysis during training. Using the datasets as input, the machine learning software trains itself to arrive at the predefined result. The software distinguishes between various word arrangements and ascertains how they impact the ultimate sentiment score through extensive training with a wide range of diverse instances.

.
Pros and cons
The accuracy with which machine learning sentiment analysis handles a wide variety of text data is one of its benefits. When enough examples are used to train the machine, ML sentiment analysis can reliably anticipate the messages' emotional tones. A taught machine learning model, however, is limited to a single industry. This implies that social media monitoring software trained on sentiment analysis using marketing data cannot be used without retraining.
Hybrid.

Rule-based methods and machine learning are used to provide hybrid sentiment analysis. It maximizes speed and accuracy when determining contextual intent in text by combining elements from both approaches. However, integrating the two disparate systems requires time and technological work.

What are the different types of sentiment analysis?
Enterprises employ diverse forms of sentiment analysis to comprehend the emotions of their clientele throughout interactions with goods or services
.
Fine-grained scoring:
Fine-grained sentiment analysis is the process of classifying text intent into distinct emotional levels. Typically, the procedure involves assigning a number to each user's feeling on a scale from 0 to 100, with equal segments denoting extremely positive, positive, neutral, negative, and very negative emotions. E-commerce sites employ a 5-star rating system as a comprehensive grading system to evaluate the client experience during a purchase.

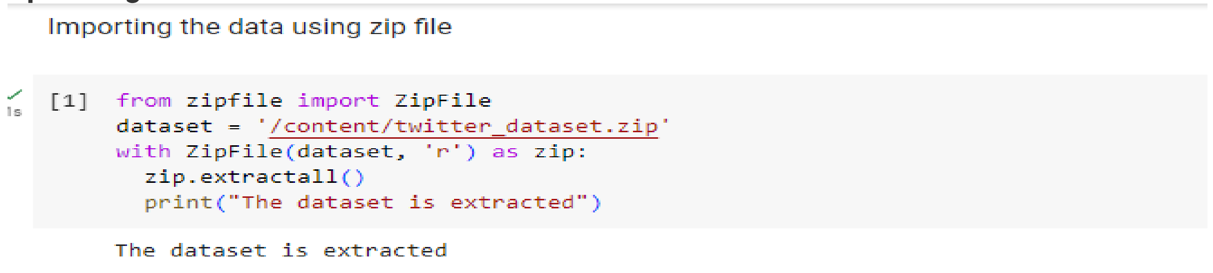
Aspect-oriented
Aspect-based analysis focuses on particular aspects of a product or service. For example, laptop manufacturers survey users regarding their satisfaction with the touchpad, keyboard, sound, and graphics. They use sentiment analysis techniques to link phrases related to hardware with customer intent.

Intent-based
Intent-based analysis helps market researchers understand customer sentiment. Marketers use opinion mining as a strategy to ascertain a target audience's position in the purchasing cycle. They identify terms like sales, discounts, and reviews from the conversations they follow and then use those terms to target potential customers with tailored adverts.

Emotional detection
Emotional detection is the study of a person's psychological condition during the composition process. Sentiment analysis is a more complex field than basic categorization for emotional detection. This technique infers a range of emotions, such as happiness, anger, sadness, and regret, from a person's word choice using sentiment analysis models.

Steps performed.:

Uploading the data.



Importing the data using zip file

```
[1] from zipfile import ZipFile
    dataset = '/content/twitter_dataset.zip'
    with ZipFile(dataset, 'r') as zip:
      zip.extractall()
      print("The dataset is extracted")
```

```
The dataset is extracted
```

Installing the sikit-learn package as it was not present already

```
[14] !pip install scikit-learn==0.13
```

```
Collecting scikit-learn==0.13
  Using cached scikit-learn-0.13.tar.gz (3.5 MB)
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: scikit-learn
```

Importing the necessary libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import re    # For regular expressions
from nltk.corpus import stopwords  # nltk is natural language toolkit
from nltk.stem.porter import PorterStemmer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

Downloading the stop words

A language's stop words are a list of terms that are often used. English stop words include "a," "the," "is," "are," and so forth. In text mining and natural language processing (NLP), stop words are frequently used to weed out terms that are so frequently used that they don't convey much meaningful information.

.

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
```

List of stop words in English:

```
print(stopwords.words('english'))
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it
```

Data processing:
Naming the columns to read the dataset again.

```
# Naming the columns and reading the dataset again
column_names = ['target','id','date','flag','user','text']
twitter_data = pd.read_csv ("/content/archive (2)/twitter_dataset..csv", names=column_names, encoding='ISO-8859-1')
```

```
twitter_data.head()
```

	target	id	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....

Shape of the dataset

```
twitter_data.shape
```

```
(1048505, 6)
```

Duplicate values

While working on a real world dataset, we might come across very messy data which involves a lot of duplicate values. Such records do not add any value or information while using them in a model and would rather slow down the processing. So, it is better to remove duplicates before feeding the data to the model
.

```
twitter_data.duplicated().sum()
```

```
73
```

```
twitter_data = twitter_data.drop_duplicates()
```

```
twitter_data.duplicated().sum()
```

```
0
```

Performing Stemming:

The goal of stemming is to simplify and standardize words, which improves the efficiency of natural language processing jobs. Stemming is a text processing technique that removes prefixes and suffixes from words, converting them into their fundamental or root form. The article delves deeper into the stemming method and demonstrates how to use it in Python.

What is Stemming in NLP?

Stemming is the process of reducing words to their most basic form; stemmers and stemming algorithms make this process easier. For instance, "retrieval" becomes "retrieve," and "chocolates" becomes "chocolate." This is important because natural language processing pipelines require tokenized words that are obtained from the initial step of breaking down a document into its individual words.

```
port_stem = PorterStemmer()
def stemming(content):
  stemmed_content = re.sub('[^a-zA-Z]',' ', content) # Remove anything which is dows not belong to a-z and A-Z
  stemmed_content = stemmed_content.lower() # Converting all the text to lower string
  stemmed_content = stemmed_content.split() # Split the words and add to the list
  stemmed_content = [port_stem.stem(word) for word in stemmed_content if not word in stopwords.words('english')] # To reduce the word to root word. And removing the stop words.
  stemmed_content = ' '.join(stemmed_content) # Join the words because they were splitted.

  return stemmed_content
```

```
twitter_data['stemmed_content'] = twitter_data['text'].apply(stemming)
```

Output after stemming:

```
twitter_data.head()
```

	target	id	date	flag	user	text	stemmed_content
0	0	1467810369	Mon Apr 06 22:19:45 PDT 2009	NO_QUERY	_TheSpecialOne_	@switchfoot http://twitpic.com/2y1zl - Awww, t...	switchfoot http twitpic com zl awww bummer sho...
1	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...	upset updat facebook text might cri result sch...
2	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...	kenichan dive mani time ball manag save rest g...
3	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire	whole bodi feel itchi like fire
4	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....	nationwideclass behav mad see

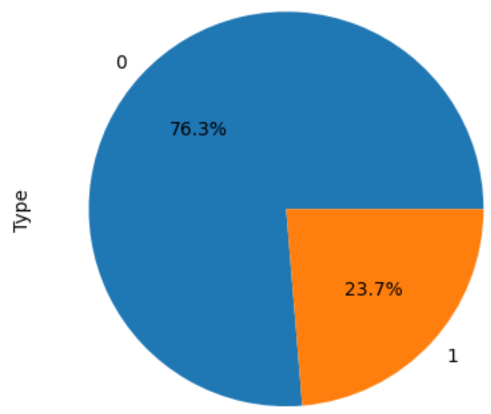
Dropping the unwanted columns as it is not required for the analysis.

```
twitter_data = twitter_data.drop(['id', 'date', 'flag', 'user', 'text'], axis=1) # Removing the unwanted columns
```

The target value us skewed because the 0 and 1 values are not equal.

```
twitter_data.groupby('target').size().plot(kind='pie',
                                            y = "target",
                                            label = "Type",
                                            autopct='%1.1f%%')
```

```
<Axes: ylabel='Type'>
```



Hence we need to perform upscaling of the values.

For upscaling series of steps are performed:

One technique for increasing the number of instances in a dataset is up sampling. It is frequently applied in situations where the data distribution is unbalanced and one class is noticeably underrepresented. Upsampling entails creating artificial data points and appending them to the dataset in an identical manner to the current data.

Duplicating already-existing instances in the underrepresented class is one method of upsampling. On the other hand, this method may cause overfitting and not provide much new information to the dataset. Alternatively, new instances that are comparable to yet distinct from the old ones can be produced using more sophisticated methods like generative models or synthetic data generation.

Creating a copy of the data frame in case of backup and splitting the data into "X_test" and "Y_test".

```
df = twitter_data.copy()
```

```
X_test = df['stemmed_content']
Y_test = df['target']
```

Because 1 is in minority and 0 is in majority, we are assigning the majority and minority class.

```
#Seperate the majority and minority classes
df_majority = df[df['target']==0]
df_minority = df[df['target']==1]
```

Combining the unsampled majority and minority class and then shuffling it.

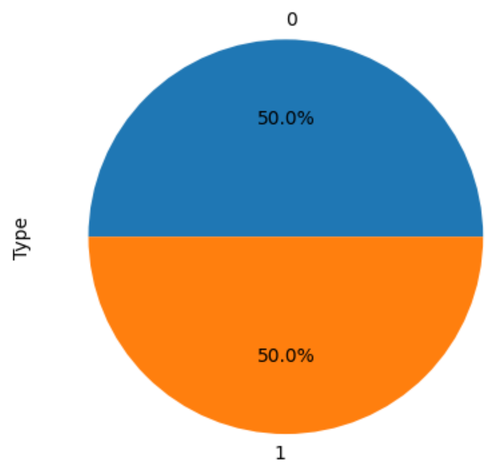
```
# Combine the upsampled minority class with the original majority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])
```

```
# Shuffle the upsampled data
df_upsampled = df_upsampled.sample(frac=1, random_state=42).reset_index(drop=True)
```

After upsampling the data, we can see the 50-50% presence of 1's and 0's

```
df_upsampled.groupby('target').size().plot(kind='pie',
                                            y = "target",
                                            label = "Type",
                                            autopct='%1.1f%%')
```

```
<Axes: ylabel='Type'>
```



Once up sampling is done we move to Model creation.

Moving the entire data to the original data frame "twitter_data"

```
twitter_data=df_upsampled.copy()
```

Splitting the data into training and testing.

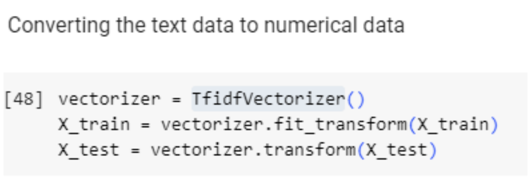
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, stratify=Y, random_state=2)
```

```
print(X.shape, X_train.shape, X_test.shape)
```

```
(1599976,) (1279980,) (319996,)
```

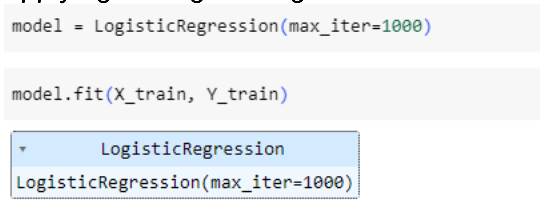
<u>Using TFIdfVectorizer:</u>

TF-IDF is an abbreviation for Term Frequency Inverse Document Frequency. This is very common algorithm to transform text into a meaningful representation of numbers which is used to fit machine algorithm.


Converting the text data to numerical data

```
[48] vectorizer = TfidfVectorizer()
     X_train = vectorizer.fit_transform(X_train)
     X_test = vectorizer.transform(X_test)
```

Applying the logistic regression the test and train data.

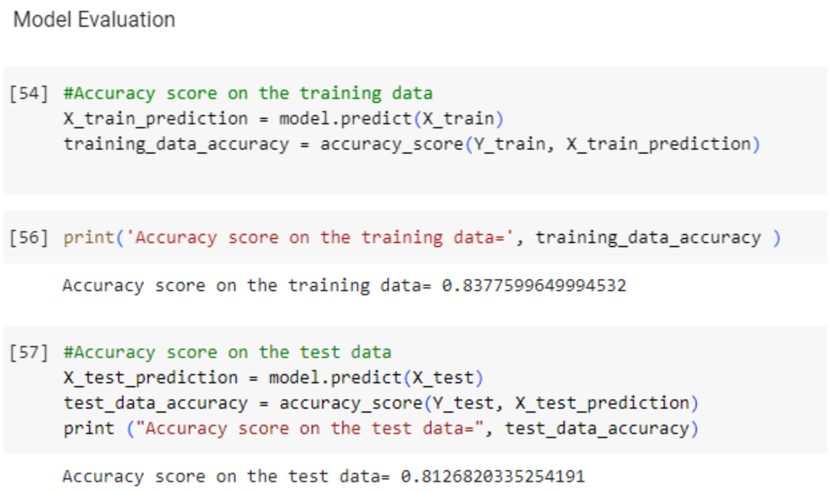


```
model = LogisticRegression(max_iter=1000)
```

```
model.fit(X_train, Y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

Model evaluation


Model Evaluation

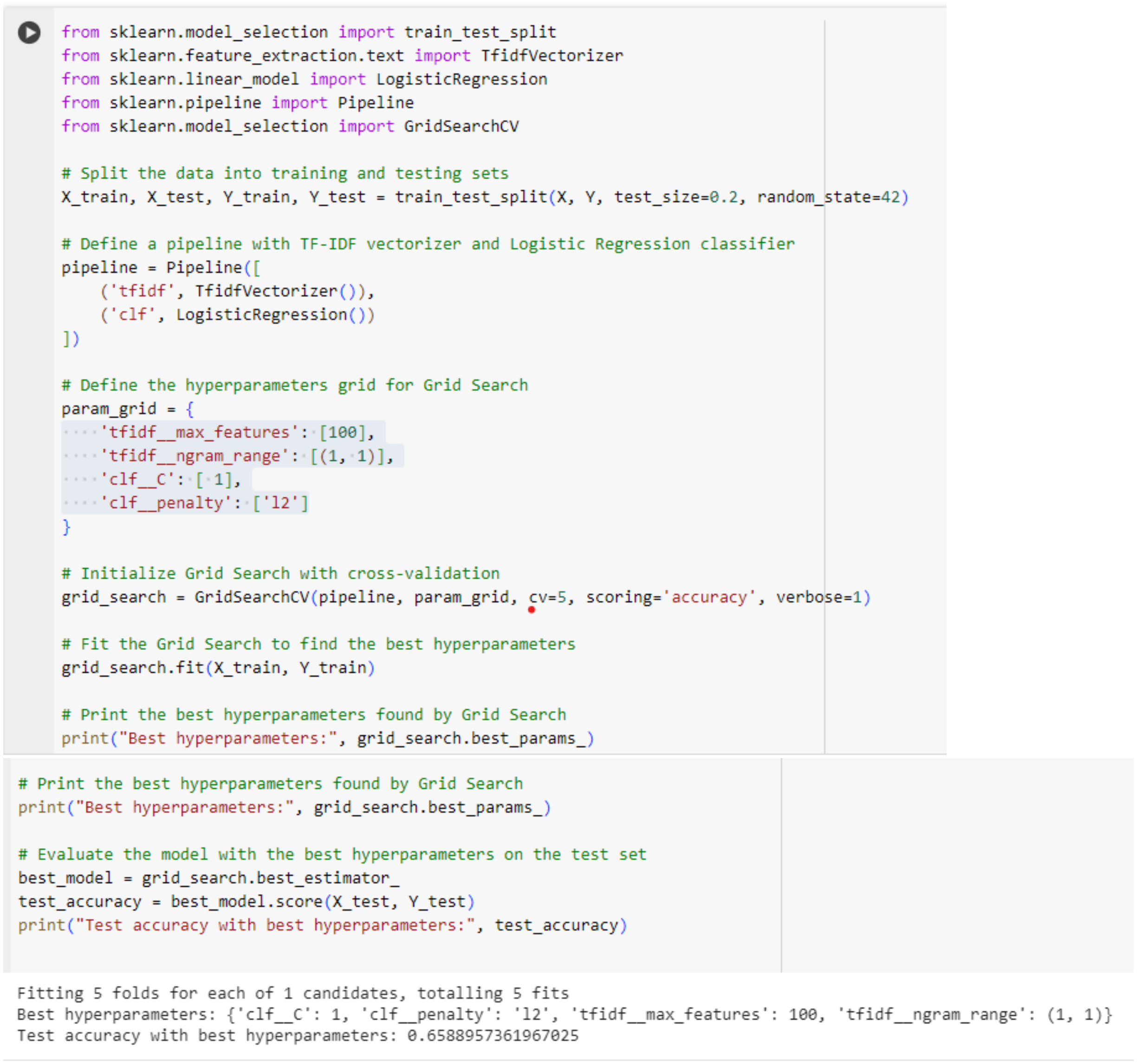
```
[54] #Accuracy score on the training data
     X_train_prediction = model.predict(X_train)
     training_data_accuracy = accuracy_score(Y_train, X_train_prediction)
```

```
[56] print('Accuracy score on the training data=', training_data_accuracy )

     Accuracy score on the training data= 0.8377599649994532
```

```
[57] #Accuracy score on the test data
     X_test_prediction = model.predict(X_test)
     test_data_accuracy = accuracy_score(Y_test, X_test_prediction)
     print ("Accuracy score on the test data=", test_data_accuracy)

     Accuracy score on the test data= 0.8126820335254191
```

Hyperparameter tuning:

Hyperparameter

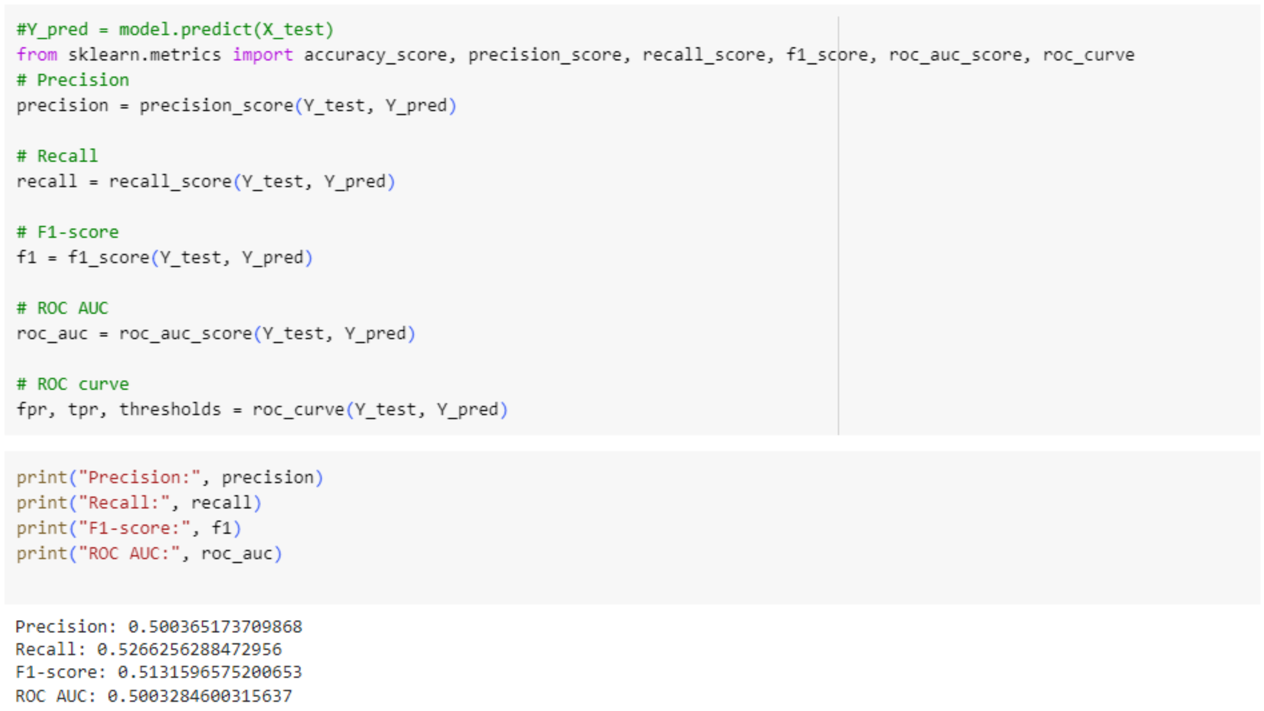


```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Define a pipeline with TF-IDF vectorizer and Logistic Regression classifier
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression())
])

# Define the hyperparameters grid for Grid Search
param_grid = {
    'tfidf__max_features': [100],
    'tfidf__ngram_range': [(1, 1)],
    'clf__C': [ 1],
    'clf__penalty': ['l2']
}

# Initialize Grid Search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', verbose=1)

# Fit the Grid Search to find the best hyperparameters
grid_search.fit(X_train, Y_train)

# Print the best hyperparameters found by Grid Search
print("Best hyperparameters:", grid_search.best_params_)
```

```
# Print the best hyperparameters found by Grid Search
print("Best hyperparameters:", grid_search.best_params_)
```

```
# Evaluate the model with the best hyperparameters on the test set
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, Y_test)
print("Test accuracy with best hyperparameters:", test_accuracy)
```

```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
Best hyperparameters: {'clf__C': 1, 'clf__penalty': 'l2', 'tfidf__max_features': 100, 'tfidf__ngram_range': (1, 1)}
Test accuracy with best hyperparameters: 0.6588957361967025
```



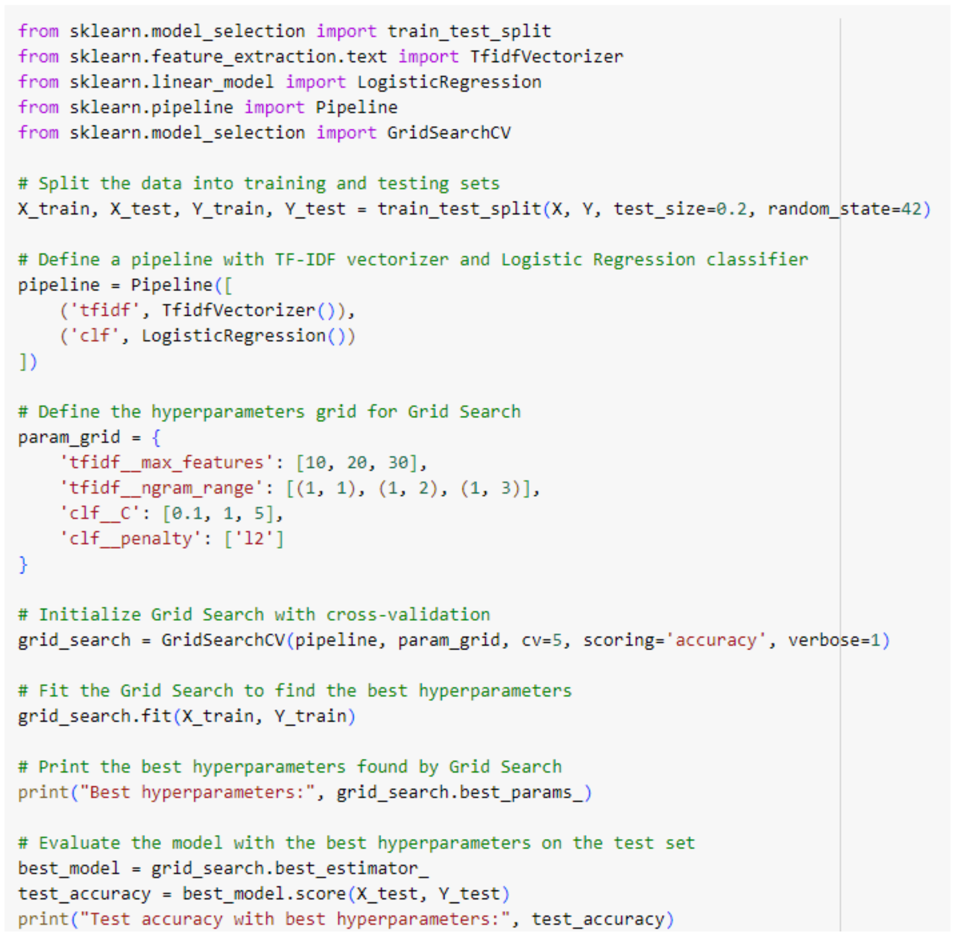
```python
#Y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
# Precision
precision = precision_score(Y_test, Y_pred)

# Recall
recall = recall_score(Y_test, Y_pred)

# F1-score
f1 = f1_score(Y_test, Y_pred)

# ROC AUC
roc_auc = roc_auc_score(Y_test, Y_pred)

# ROC curve
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred)
```

```python
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC AUC:", roc_auc)
```

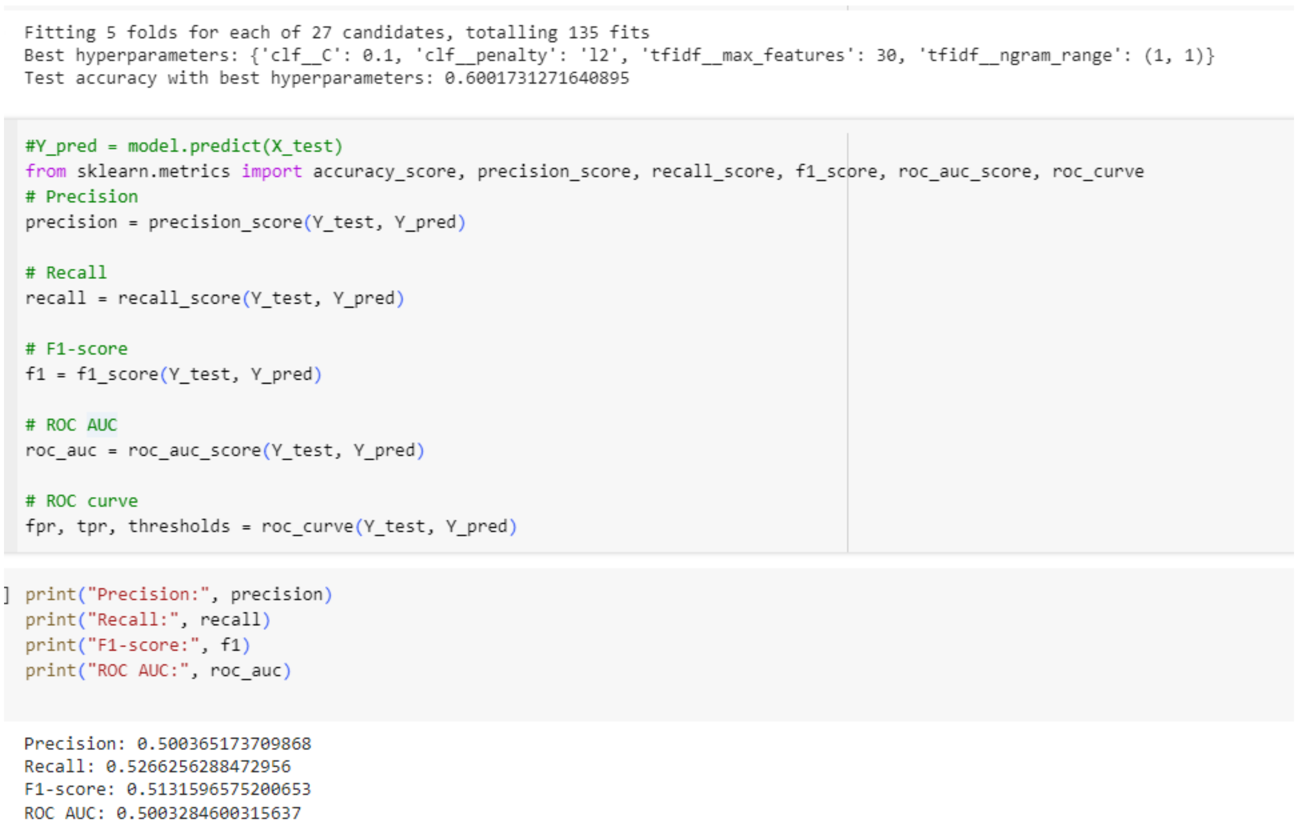
```
Precision: 0.500365173709868
Recall: 0.5266256288472956
F1-score: 0.5131596575200653
ROC AUC: 0.5003284600315637
```

Conclusion for hyperparameter tuning:

We tried different parameters but the accuracy without hyperparameter was better.



```python
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV

# Split the data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

# Define a pipeline with TF-IDF vectorizer and Logistic Regression classifier
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression())
])

# Define the hyperparameters grid for Grid Search
param_grid = {
    'tfidf__max_features': [10, 20, 30],
    'tfidf__ngram_range': [(1, 1), (1, 2), (1, 3)],
    'clf__C': [0.1, 1, 5],
    'clf__penalty': ['l2']
}

# Initialize Grid Search with cross-validation
grid_search = GridSearchCV(pipeline, param_grid, cv=5, scoring='accuracy', verbose=1)

# Fit the Grid Search to find the best hyperparameters
grid_search.fit(X_train, Y_train)

# Print the best hyperparameters found by Grid Search
print("Best hyperparameters:", grid_search.best_params_)

# Evaluate the model with the best hyperparameters on the test set
best_model = grid_search.best_estimator_
test_accuracy = best_model.score(X_test, Y_test)
print("Test accuracy with best hyperparameters:", test_accuracy)
```



```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Best hyperparameters: {'clf__C': 0.1, 'clf__penalty': 'l2', 'tfidf__max_features': 30, 'tfidf__ngram_range': (1, 1)}
Test accuracy with best hyperparameters: 0.6001731271640895
```

```
#Y_pred = model.predict(X_test)
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, roc_curve
# Precision
precision = precision_score(Y_test, Y_pred)

# Recall
recall = recall_score(Y_test, Y_pred)

# F1-score
f1 = f1_score(Y_test, Y_pred)

# ROC AUC
roc_auc = roc_auc_score(Y_test, Y_pred)

# ROC curve
fpr, tpr, thresholds = roc_curve(Y_test, Y_pred)
```

```
] print("Precision:", precision)
  print("Recall:", recall)
  print("F1-score:", f1)
  print("ROC AUC:", roc_auc)
```

```
Precision: 0.500365173709868
Recall: 0.5266256288472956
F1-score: 0.5131596575200653
ROC AUC: 0.5003284600315637
```

Reference:- https://www.kaggle.com/datasets/kazanova/sentiment140/code

https://www.cse.ust.hk/~rossiter/independent_studies_projects/twitter_emotion_analysis/twitter_emotion_analysis.pdf

https://aws.amazon.com/what-is/sentiment-analysis/

https://medium.com/@rithpansanga/choosing-the-right-size-a-look-at-the-differences-between-upsampling-and-downsampling-methods-daae83915c19

https://medium.com/@cmukesh8688/tf-idf-vectorizer-scikit-learn-dbc0244a911a