MACHINE LEARNING

(Twitter Sentiment Analysis)

Summer Internship Report Submitted in partial fulfillment of the

requirement for undergraduate degree of

Bachelor of Technology

In

Computer Science Engineering

By

Shaik Shoaib Aslam

221710302060

Under the Guidance of

Assistant Professor

Department Of Computer Science Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

DECLARATION

I submit this industrial training work entitled "**TWITTER SENTIMENT ANALYSIS**" to GITAM (Deemed To Be University), Hyderabad in partial fulfillment of the requirements for the award of the degree of "Bachelor of Technology" in "Computer Science Engineering". I declare that it was carried out independently by me under the guidance of , Asst. Professor, GITAM (Deemed To Be University), Hyderabad, India.

The results embodied in this report have not been submitted to any other University or Institute for the award of any degree or diploma.

Place: HYDERABAD — Shaik Shoaib Aslam

Date: 14 July 2020 — 221710302060

GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled "**TWITTER SENTIMENT ANALYSIS**" is being submitted by SHAIK SHOAIB ASLAM (221710302060) in partial fulfillment of the requirement for the award of Bachelor of Technology in Computer Science Engineering at GITAM (Deemed To Be University), Hyderabad during the academic year 2019-20.

It is faithful record work carried out by him at the Computer Science Engineering Department, GITAM University Hyderabad Campus under my guidance and supervision.

	Dr. S Phani Kumar
Assistant Professor	Professor and HOD
Department of CSE	Department of CSE

ACKNOWLEDGEMENT

Apart from my effort, the success of this internship largely depends on the encouragement and guidance of many others. I take this opportunity to express my gratitude to the people who have helped me in the successful competition of this Internship.

I would like to thank Dr. N. Siva Prasad, Pro Vice Chancellor, GITAM Hyderabad and N. Seetharamaiah, Principal, GITAM Hyderabad.

I would like to thank respected Dr. S. Phani Kumar, Head of the Department of Computer Science Engineering for giving me such a wonderful opportunity to expand my knowledge for my own branch and giving me guidelines to present an internship report. It helped me a lot to realize what we study for.

I would like to thank the respected faculties who helped me to make this internship a successful accomplishment.

I would also like to thank my friends who helped me to make my work more organized and well-stacked till the end.

Shaik Shoaib Aslam

221710302060

ABSTRACT

Machine learning algorithms are used for predictions on various real-time applications like pattern recognition,sentiment analysis, these algorithms are trained on a particular dataset and then tested for different datasets/unseen datasets. And a particular evaluation metric like accuracy score or f1-score is considered(in this case study accuracy score).The models are measured based on these evaluation metrics and further implementation is done. In this Twitter Sentiment Analysis case-study we are trying to take the input from the client about any trending hashtag on Twitter. And display whether the tweets are of negative or positive category. My primary objective in this case study is to check what percent of that received tag are negative and what percent of them are positive.

Table of Contents

LIST OF FIGURES

CHAPTER 1

MACHINE LEARNING

1.1 INTRODUCTION:

Machine Learning(ML) is the scientific study of algorithms and statistical models that computer systems use in order to perform a specific task effectively without using explicit instructions, relying on patterns and inference instead. It is seen as a subset of Artificial Intelligence(AI).

1.2 IMPORTANCE OF MACHINE LEARNING:

Consider some of the instances where machine learning is applied: the self-driving Google car, cyber fraud detection, online recommendation engines—like friend suggestions on Facebook, Netflix showcasing the movies and shows you might like, and “more items to consider” and “get yourself a little something” on Amazon—are all examples of applied machine learning. All these examples echo the vital role machine learning has begun to take in today’s data-rich world.

Machines can aid in filtering useful pieces of information that help in major advancements, and we are already seeing how this technology is being implemented in a wide variety of industries.

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that’s in part due to increased sophistication of machine learning, which helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques

The process flow depicted here represents how machine learning works

Figure 1.2.1 : The Process Flow

1.3 USES OF MACHINE LEARNING:

Earlier in this article, we mentioned some applications of machine learning. To understand the concept of machine learning better, let's consider some more examples: web search results, real-time ads on web pages and mobile devices, email spam filtering, network intrusion detection, and pattern and image recognition. All these are by-products of applying machine learning to analyze huge volumes of data

Traditionally, data analysis was always being characterized by trial and error, an approach that becomes impossible when data sets are large and heterogeneous. Machine learning comes as the solution to all this chaos by proposing clever alternatives to analyzing huge volumes of data. By developing fast and efficient algorithms and data-driven models for real-time processing of data, machine learning can produce accurate results and analysis.

1.4 TYPES OF LEARNING ALGORITHMS:

The types of machine learning algorithms differ in their approach, the type of data they input and output, and the type of task or problem that they are intended to solve.

1.4.1 Supervised Learning :

When an algorithm learns from example data and associated target responses that can consist of numeric values or string labels, such as classes or tags, in order to later predict the correct response when posed with new examples comes under the category of supervised learning.

Supervised machine learning algorithms uncover insights, patterns, and relationships from a labelled training dataset – that is, a dataset that already contains a known value for the target variable for each record. Because you provide the machine learning algorithm with the correct answers for a problem during training, it is able to "learn" how the rest of the features relate to the target, enabling you to uncover insights and make predictions about future outcomes based on historical data.

Examples of Supervised Machine Learning Techniques are Regression, in which the algorithm returns a numerical target for each example, such as how much revenue will be generated from a new marketing campaign.

Classification, in which the algorithm attempts to label each example by choosing between two or more different classes. Choosing between two classes is called binary classification, such as determining whether or not someone will default on a loan. Choosing between more than two classes is referred to as multiclass classification.

1.4.2 Unsupervised Learning:

When an algorithm learns from plain examples without any associated response, leaving to the algorithm to determine the data patterns on its own. This type of algorithm tends to restructure the data into something else, such as new features that may represent a class or a new series of uncorrelated values. They are quite useful in providing humans with insights into the meaning of data and new useful inputs to supervised machine learning algorithms.

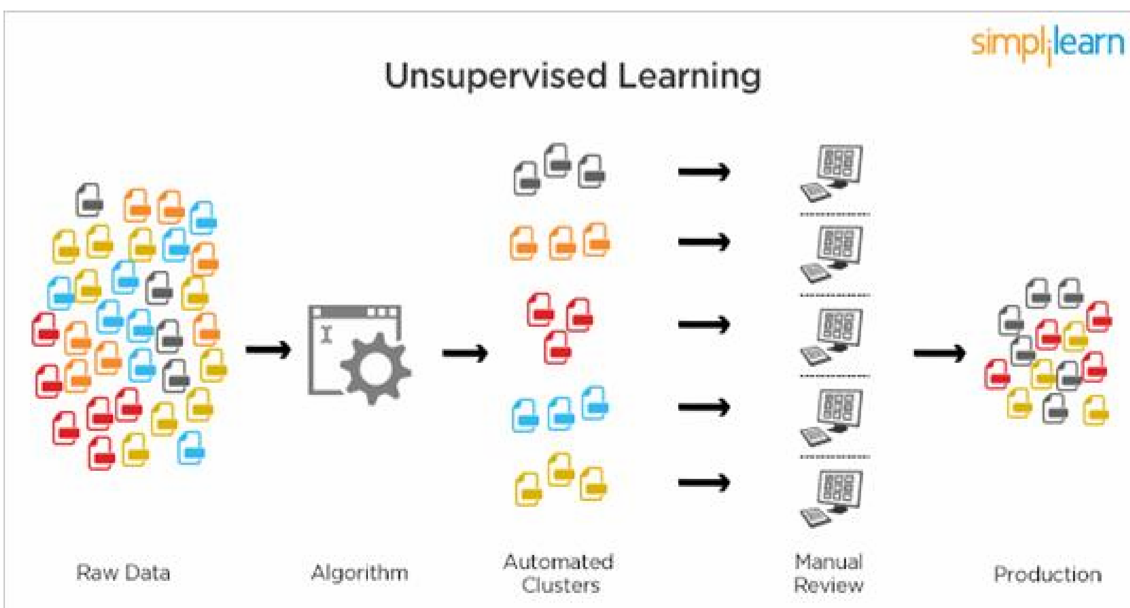


Figure 1.4.2.1 : Unsupervised Learning

Popular techniques where unsupervised learning is used also include self-organizing maps, nearest neighbor mapping, singular value decomposition, and k-means clustering. Basically, online recommendations, identification of data outliers, and segment text topics are all examples of unsupervised learning.

1.4.3 Semi Supervised Learning:

As the name suggests, semi-supervised learning is a bit of both supervised and unsupervised learning and uses both labeled and unlabeled data for training. In a typical scenario, the algorithm would use a small amount of labeled data with a large amount of unlabeled data.

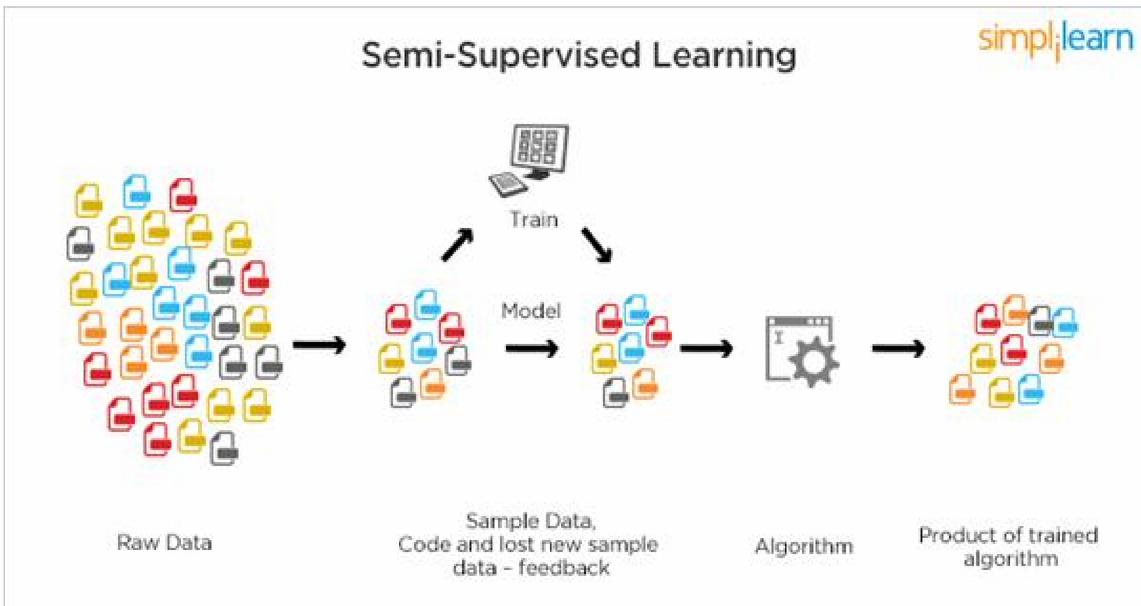


Figure 1.4.3.1 : Semi Supervised Learning

1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING

Machine learning and data mining use the same algorithms and techniques as data mining, except the kinds of predictions vary. While data mining discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

Deep learning, on the other hand, uses advanced computing power and special types of neural networks and applies them to large amounts of data to learn, understand, and identify complicated patterns. Automatic language translation and medical diagnoses are examples of deep learning.

CHAPTER 2

PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYTHON:

- Python is a high-level, interpreted, interactive and object-oriented scripting language.
- Python is a general purpose programming language that is often applied in scripting roles
- Python is Interpreted: Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is like PERL and PHP.
- Python is Interactive: You can sit at a Python prompt and interact with the interpreter directly to write your programs.

- Python is Object-Oriented: Python supports the Object-Oriented style or technique of programming that encapsulates code within objects.

2.2 HISTORY OF PYTHON:

- Python was developed by GUIDO VAN ROSSUM in early 1990’s
- Its latest version is 3.7 , it is generally called as python3

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes.
- Easy-to-maintain: Python's source code is fairly easy-to-maintaining.
- A broad standard library: Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- Portable: Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- Extendable: You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- Databases: Python provides interfaces to all major commercial databases.
- GUI Programming: Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

2.4 HOW TO SETUP PYTHON

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.

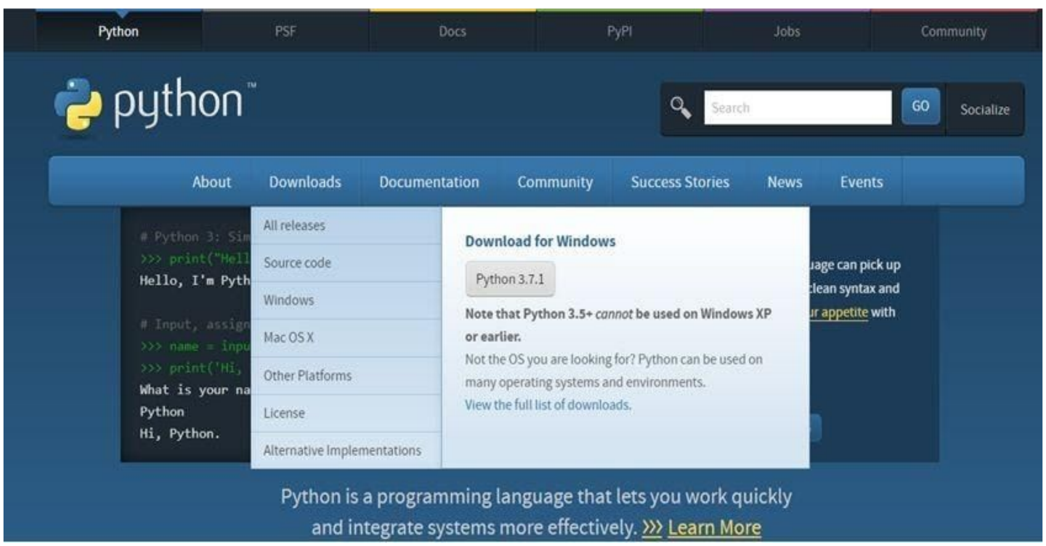


Figure 2.4.1 : Python download

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.
- Anaconda is a free open source distribution of python for large scale data processing, predictive analytics and scientific computing.
- Conda is a package manager that quickly installs and manages packages.

In WINDOWS:

- In windows
  - Step 1: Open Anaconda.com/downloads in a web browser.
  - Step 2: Download python 3.4 version for (32-bits graphic installer/64 -bit graphic installer)
  - Step 3: select installation type(all users)
  - Step 4: Select path(i.e. add anaconda to path & register anaconda as default python 3.4) next click install and next click finish
  - Step 5: Open jupyter notebook (it opens in default browser)

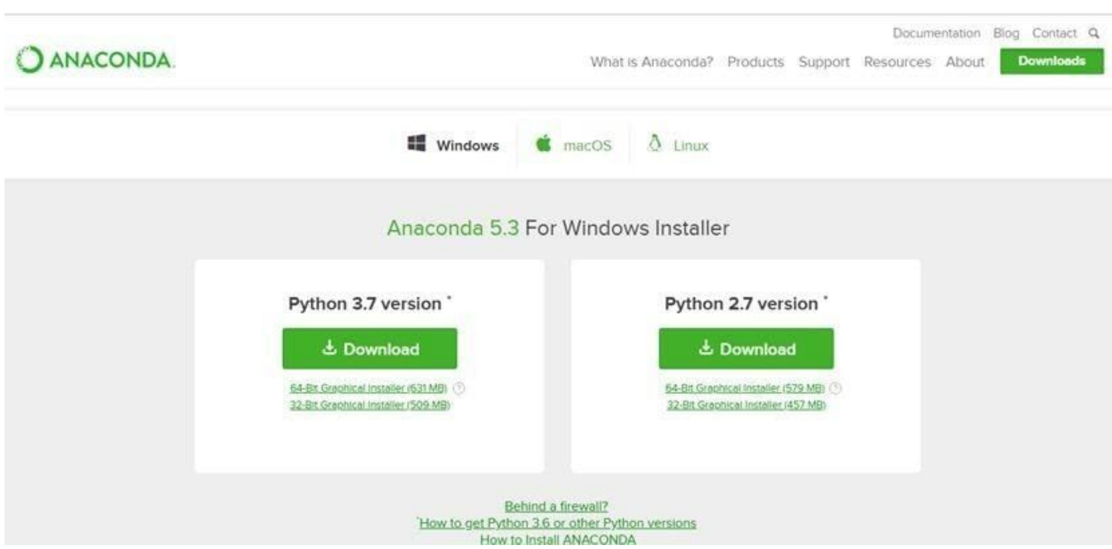


Figure 2.4.2.1: Anaconda download

Figure 2.4.2.2 : Jupyter notebook

2.5 PYTHON VARIABLE TYPES:

- Variables are nothing but reserved memory locations to store values. This means that when you create a variable you reserve some space in memory.
- Variables are nothing but reserved memory locations to store values.
- Based on the data type of a variable, the interpreter allocates memory and decides what can be stored in the reserved memory.
- Python variables do not need explicit declaration to reserve memory space. The declaration happens automatically when you assign a value to a variable.
- Python has various standard data types that are used to define the operations possible on them and the storage method for each of them.
- Python has five standard data types –
    - Numbers
    - Strings
    - Lists
    - Tuples
    - Dictionary

2.5.1 Python Numbers:

- Number data types store numeric values. Number objects are created when you assign a value to them.

- Python supports four different numerical types − int (signed integers) long (long integers, they can also be represented in octal and hexadecimal) float (floating point real values) complex (complex numbers).

2.5.2 Python Strings:

- Strings in Python are identified as a contiguous set of characters represented in the quotation marks.

- Python allows for either pairs of single or double quotes.
- Subsets of strings can be taken using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

- The plus (+) sign is the string concatenation operator and the asterisk (*) is the repetition operator.

2.5.3 Python Lists:

- Lists are the most versatile of Python's compound data types.

- A list contains items separated by commas and enclosed within square brackets-([]).
- To some extent, lists are similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

- The values stored in a list can be accessed using the slice operator ([ ] and [:]) with indexes starting at 0 in the beginning of the list and working their way to end -1.

- The plus (+) sign is the list concatenation operator, and the asterisk (*) is the repetition operator.

2.5.4 Python Tuples:

- A tuple is another sequence data type that is similar to the list.
- A tuple consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.
- The main differences between lists and tuples are: Lists are enclosed in brackets ([ ]) and their elements and size can be changed, while tuples are enclosed in parentheses (( )) and cannot be updated.
- Tuples can be thought of as read-only lists.
- For example − Tuples are fixed size in nature whereas lists are dynamic. In other words, a tuple is immutable whereas a list is mutable. You can't add elements to a tuple. Tuples have no append or extend method. You can't remove elements from a tuple. Tuples have no remove or pop method.

2.5.5 Python Dictionary:

- Python's dictionaries are kind of hash table type. They work like associative arrays
- or hashes found in Perl and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object.
- Dictionaries are enclosed by curly braces ({ }) and values can be assigned and accessed using square braces ([]).
- You can use numbers to "index" into a list, meaning you can use numbers to find out what's in lists. You should know this about lists by now, but make sure you understand that you can only use numbers to get items out of a list.
- What a dict does is let you use anything, not just numbers. Yes, a dict associates one thing to another, no matter what it is.

2.6 PYTHON FUNCTION

2.6.1 Defining a Function:

You can define functions to provide the required functionality. Here are simple rules to define a function in Python. Function blocks begin with the keyword def followed by the function name and parentheses (i.e.()).

Any input parameters or arguments should be placed within these parentheses.
You can also define parameters inside these parentheses

The code block within every function starts with a colon (:) and is indented. The statement returns [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

2.6.2 Calling a Function:

Defining a function only gives it a name, specifies the parameters that are to be included in the function and structures the blocks of code. Once the basic structure of a function is finalized, you can execute it by calling it from another function or directly from the Python prompt.

2.7 PYTHON USING OOPs CONCEPTS:

2.7.1 Class:

- Class: A user-defined prototype for an object that defines a set of attributes that characterize any object of the class. The attributes are data members (class variables and instance variables) and methods, accessed via dot notation.

- Class variable: A variable that is shared by all instances of a class. Class variables are defined within a class but outside any of the class's methods. Class variables are not used as frequently as instance variables are.

- Data member: A class variable or instance variable that holds data associated with a class and its objects.

- Instance variable: A variable that is defined inside a method and belongs only to the current instance of a class.

- Defining a Class:
    - o We define a class in a very similar way how we define a function.
    - o Just like a function ,we use parentheses and a colon after the class name(i.e. ():) when we define a class. Similarly, the body of our class is indented like a functions body is.

```
def my_function():
    # the details of the
    # function go here
```

```
class MyClass():
    # the details of the
    # class go here
```

Figure 2.7.1 : Defining a Class

2.7.2 init method in Class:

- The init method — also called a constructor — is a special method that runs when an instance is created so we can perform any tasks to set up the instance.

- The init method has a special name that starts and ends with two underscores: init ().

CHAPTER 3
TWITTER SENTIMENT ANALYSIS

3.1 PROBLEM STATEMENT

To predict the sentiments for Twitter data using Machine Learning Algorithms LOGISTIC REGRESSION and DECISION TREE.

3.2 DATASET

1. Id
2. Label - 0/1 (0 for Positive and 1 for Negative)
3. tweet

3.3 OBJECTIVE OF THE CASE STUDY

The primary objective of this project is to predict the Twitter data, whether it is positive or negative

CHAPTER 4: DATA PREPROCESSING/FEATURE ENGINEERING

4.1 READING DATA

Preparing text data using following steps:

4.1.1 Getting the dataset

We can get the twitter handling data by scraping the web

4.1.2 Importing Required Libraries

```
[ ]  # Importing Required libraries
     import pandas as pd
     import numpy as np
     import seaborn as sns
     import re
```

Figure 4.1.2.1 : Importing Libraries

4.1.3 Import Data

Reading Training data

```
[ ]  # Reading the dataset
     tweets=pd.read_csv("/content/drive/My Drive/2020/train.csv",encoding = 'latin - 1')
     tweets.head()
```

	id	label	tweet
0	1	0	@user when a father is dysfunctional and is s...
1	2	0	@user @user thanks for #lyft credit i can't us...
2	3	0	bihday your majesty
3	4	0	#model i love u take with u all the time in ...
4	5	0	factsguide: society now #motivation

Figure 4.1.3: Reading data and print head

4.2 STATISTICAL ANALYSIS

4.2.1 Checking Shape of dataset

```
[ ] # Checking size of dataset
    tweets.shape

    (31962, 3)
```

Figure 4.2.1: Shape of Dataset

4.2.2 Checking Frequency of Output Column

```
[ ] tweets.label.value_counts()

    0    29720
    1     2242
    Name: label, dtype: int64
```

Figure 4.2.2 : label counts

Imbalance in the dataset can be seen.

4.2.3 Visualizing Output Column

Using Count plot to visualize Output column

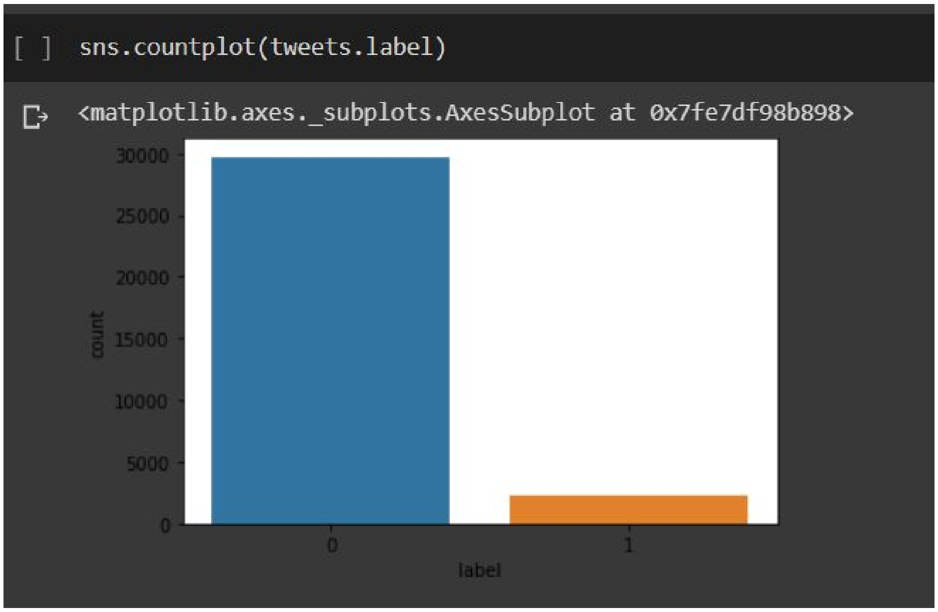


Figure 4.2.3: Visualizing label counts

From the Plot, we can see that the frequency of 0's i.e, positive values are more in number, which makes the dataset inefficient. So, we need to apply some techniques to balance the dataset.

4.2.4 Statistical Description

```
[ ] # Checking Statistical data
    tweets.describe()
```

	id	label
count	31962.000000	31962.000000
mean	15981.500000	0.070146
std	9226.778988	0.255397
min	1.000000	0.000000
25%	7991.250000	0.000000
50%	15981.500000	0.000000
75%	23971.750000	0.000000
max	31962.000000	1.000000

Figure 4.2.4: Statistical Description

4.3 HANDLING MISSING VALUES

```
[ ] # Checking for NULL values
    tweets.isna().sum() # No NULL Values can be found

    id       0
    label    0
    tweet    0
    dtype: int64
```

Figure 4.3: Checking missing values

No missing values found from the dataset.

4.4 CLEANING TEXT DATA WITH NLTK

4.4.1 Downloading Required Resources From Nltk

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

    [nltk_data] Downloading package stopwords to /root/nltk_data...
    [nltk_data]   Package stopwords is already up-to-date!
    True
```

Figure 4.4.1.1: nltk stopwords download

```
[ ] import nltk
    nltk.download('wordnet')

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

Figure 4.4.1.2: nltk wordnet download

4.4.2 Removing Stopwords

```
[ ] # Removing Stopwords
    from nltk.corpus import stopwords # Stop words contain words like a,an,the,is,are,...etc
    stop=stopwords.words("english")
    stop.extend(["i'm","I'm"]) # adding additional stopwords
    # Implementing stopwords on train data
    tweets.tweet=tweets.tweet.apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
    tweets.head()
```

	id	label	tweet
0	1	0	@user father dysfunctional selfish drags kids ...
1	2	0	@user @user thanks #lyft credit can't use caus...
2	3	0	bihday majesty
3	4	0	#model love u take u time urÃ°Â□Â□Â±!!! Ã°Â□Â□...
4	5	0	factsguide: society #motivation

Figure 4.4.2: Removing stopwords

Ex: @user Project 8 is the best car ever made! ---> @user Project 8 best car ever made!.

4.4.3 Removing Hyperlinks And Mentions

Creating a function which removes the hyperlinks and mentions from the text data using a regular expression package.

```
def clean(x):
    x=' '.join(re.sub("(@[A-Za-z0-9]+)|([^A-Za-z0-9']+)|(\w+:\/\/\S+)"," ",x).split())
    return x
```

Figure 4.4.3: Removing Hyperlinks

```
[ ] # Removing Hyperlinks, userIDS
    tweets.tweet = tweets.tweet.apply(clean)
    tweets.head()
```

	id	label	tweet
0	1	0	father dysfunctional selfish drags kids dysfun...
1	2	0	user thanks lyft credit can't use cause offer ...
2	3	0	bihday majesty
3	4	0	model love u take u time ur
4	5	0	factsguide society motivation

Figure 4.4.3: Removing Hyperlinks

Ex: @user Project 8 is the best car ever made! www.github.com #powerful ---> Project 8 best car ever made! Powerful.

4.4.4. Lemmatization

This process involves resolving words to their dictionary form.

```
# Applying Lemmatization
from nltk.stem.wordnet import WordNetLemmatizer
wnl = WordNetLemmatizer()
tweets.tweet=tweets.tweet.apply(lambda x:' '.join([wnl.lemmatize(word,'v') for word in x.split()])) # v stands for verb
tweets.head()
```

	id	label	tweet
0	1	0	father dysfunctional selfish drag kid dysfunct...
1	2	0	user thank lyft credit can't use cause offer w...
2	3	0	bihday majesty
3	4	0	model love u take u time ur
4	5	0	factsguide society motivation

Figure 4.4.4: Lemmatization

Ex : happiness , happi → happy

Stemming: A similar approach to lemmatization , it is a process of reducing inflected words to their base or stem words.Stemming is also a similar technique to lemmatization, but stemming is not as sophisticated as Lemmatization.

4.4.5 Converting To Lowercase

```
# Convert all the text data into lower case for flexibility
tweets.tweet=tweets.tweet.apply(lambda x:' '.join([word.lower() for word in x.split()]))
```

Figure 4.4.5: Lowercase conversion

4.4.6 Applying Same Techniques For Test Data For Preprocessing

```
# Reading test data
X_test = pd.read_csv('/content/drive/My Drive/2020/test.csv',encoding='latin- 1')
# Removing Stopwords
X_test.tweet=X_test.tweet.apply(lambda x: ' '.join([word for word in x.split() if word not in (stop)]))
# Removing Hyperlinks, userIDS
X_test.tweet = X_test.tweet.apply(clean)
# Applying Lemmatization
wnl1 = WordNetLemmatizer()
X_test.tweet=X_test.tweet.apply(lambda x:' '.join([wnl1.lemmatize(word,'v') for word in x.split()])) # v stands for verb
X_test.tweet=X_test.tweet.apply(lambda x:' '.join([word.lower() for word in x.split()]))
```

Figure 4.4.6: Applying all cleaning techniques to test data

4.4.7 Reading test data output

```
[ ]  y_test = pd.read_csv('/content/drive/My Drive/2020/result.csv')
```

Figure 4.4.7: Reading test data output

4.5 CREATING BAG OF WORDS

Applying TF-IDF Vectorizer

TF : Term Frequency, which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more often in long documents than shorter ones .

TF(t) = (Number of times term t appears in a document) / (Total number of terms in the document).

Inverse Document Frequency : This downscales words that appear a lot across documents , which measures how important a term is. While computing TF, all terms are considered equally important.

IDF(t) = log(Total number of documents / Number of documents with term t in it) .

The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents. Alternatively, if you already have a learned CountVectorizer, you can use it with a TfidfTransformer to just calculate the inverse document frequencies and start encoding documents.

```
## Importing TFIDF Vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf = TfidfVectorizer()
```

Figure 4.5: Tfidf Vectorizer

4.5.1 Applying to both train and test data

```
#Applying tfidf to train data
new_inp = tfidf.fit_transform(tweets.tweet)
new_inp
#Applying tfidf to test data
test_inp = tfidf.transform(X_test.tweet)
test_inp
```

```
<17197x35865 sparse matrix of type '<class 'numpy.float64'>'
        with 123657 stored elements in Compressed Sparse Row format>
```

Figure 4.5.1: Applying Tfidf Vectorizer

4.6 BALANCING DATASET

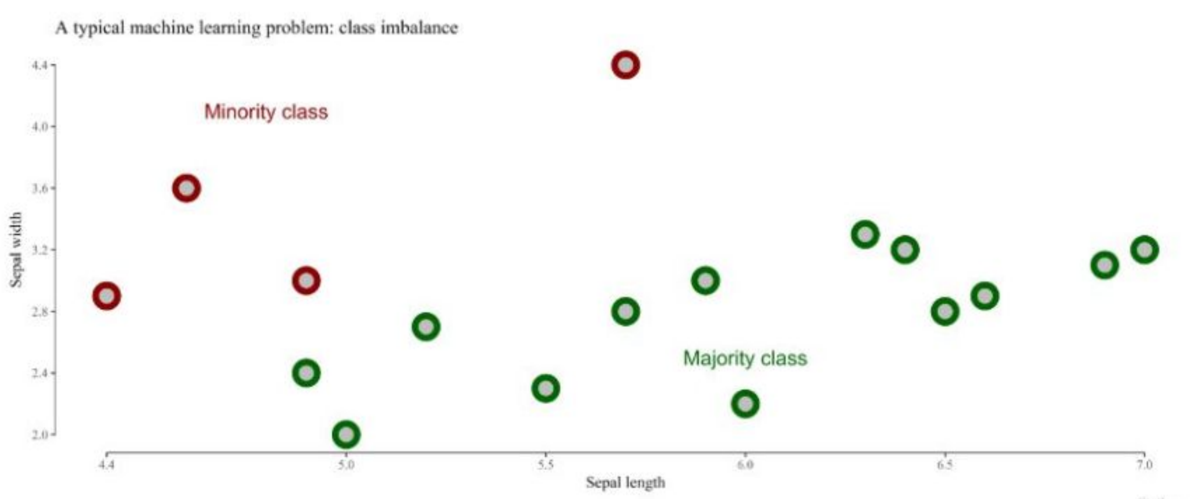
From 4.2.2 and 4.2.3 we observed that the data is imbalanced.i.e., approximately in ratio 15:1.Imbalanced data typically refers to a problem with classification problems where the classes are unequal. Imbalanced data typically refers to a classification problem where the number of observations per class is not equally distributed. Often you'll have a large amount of data/observations for one class (referred to as the majority class), and much fewer observations for one or more other classes (referred to as the minority classes.

For example, in this data we have a 2-class (binary) classification problem with instances (rows). A total of 29720 instances are labeled with Class-0 and the remaining 2242 instances are labeled with Class-1.

GENERATING SAMPLES USING SMOTE

SMOTE stands for "**Synthetic Minority Over-sampling Technique**"

This Technique synthesises new minority instances between existing (real) minority instances



From the Figure we can see the red color shows the minority data.

SMOTE synthesises new minority instances between existing (real) minority instances.

SMOTE draws lines between existing minority instances like this.

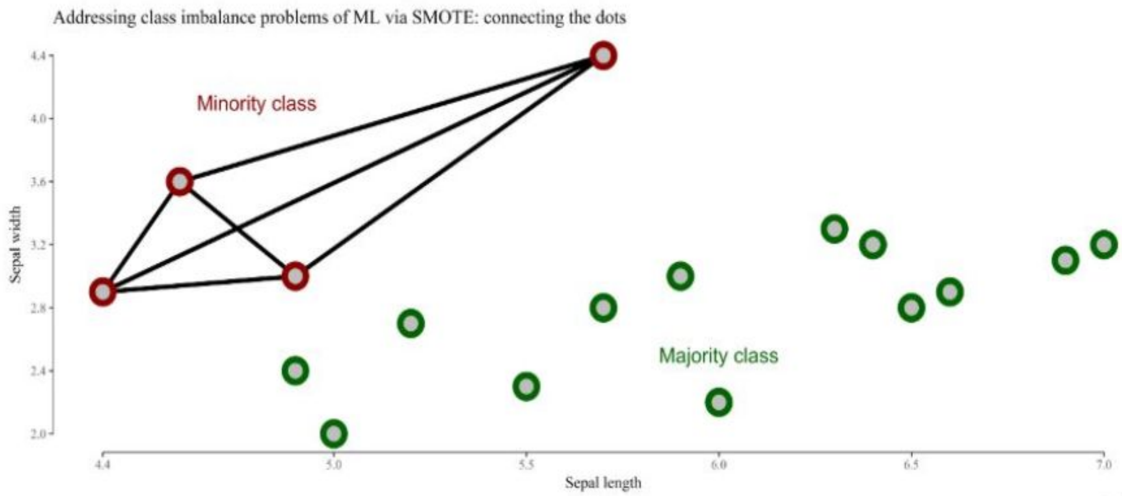


Figure 4.6: SMOTE Lines

SMOTE then imagines new, synthetic minority instances somewhere on these lines

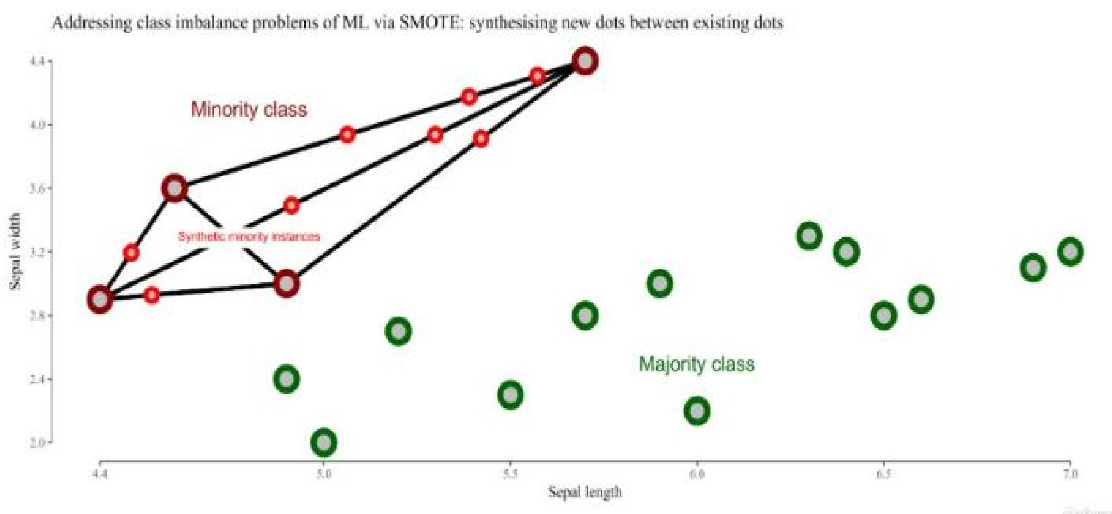


Figure 4.6.1: Synthetic data point for minority class

Importing SMOTETomek and fitting on the train data.

```
#Importing SMOTETomek
from imblearn.combine import SMOTETomek
smk = SMOTETomek(random_state=42) # Creating an Object
X_train,y_train=smk.fit_sample(new_inp,tweets.label)
```

Figure 4.6.2: Applying smote to data

The SMOTE is fitted on to the required data columns of train data

```
print(X_train.shape)
print(y_train.shape)

(59440, 35865)
(59440,)
```

Figure 4.6.3: Shape of dataset after SMOTE

Converting into dataframe to check value_counts()

```
# converting y_train from np.array to data frame to check for the balancing outcome
c=pd.DataFrame(y_train)
#  using index to generate counts of the label
c.iloc[:,0].value_counts()
# we can see that both the labels are now balanced

1    29720
0    29720
Name: 0, dtype: int64
```

Figure 4.6.4: value counts after SMOTE

We can now see that the data is balanced.

Now the data is ready to be fitted using appropriate algorithms.

CHAPTER 5
MODEL BUILDING AND EVALUATION

5.1 LOGISTIC REGRESSION

5.1.1 About Algorithm

Logistic Regression model predicts the probability associated with each dependent variable category. Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary).

The formula used in this regression is

P = e^y / (1+e^y)

This step assigns classes labels to 0 or 1 to our predicted probabilities.If p is less than 0.5, we conclude the predicted output is 0 and if p is greater than 0.5, you conclude the output is 1.

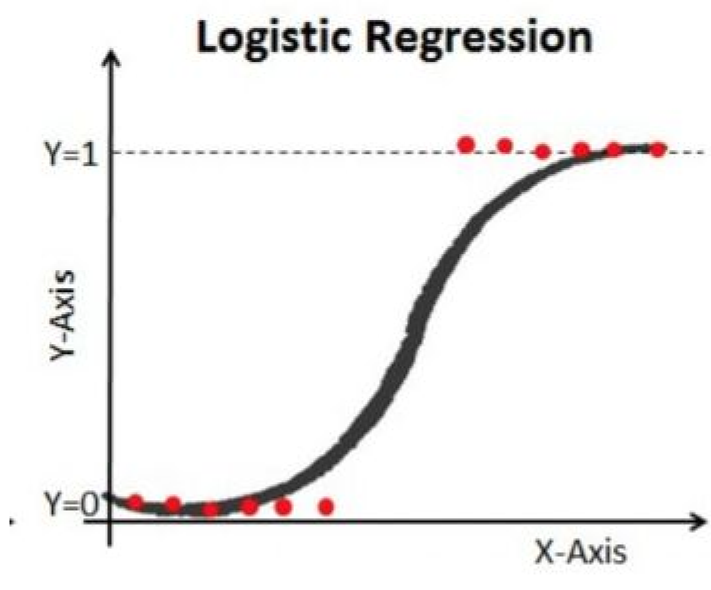


Figure 5.1.1: Graph for logistic regression

In Logistic Regression, the values always range strictly between 0 and 1.

5.1.2 Train The Model

In Machine Learning in order to access the performance of the classifier. You train the classifier using 'training set' and then test the performance of your classifier on unseen 'test set'. An important point to note is that during training the classifier only uses the training set . The test set must not be used during training the classifier. The test set will only be available during testing the classifier.training set - a subset to train a model.(Model learns patterns between Input and Output)

We need to import logistic regression method from linear_model package from scikit learn library

We need to train the model based on our train set.

1. Importing logistic regression
2. Creating Object
3. Fitting train data

```
# importing logistic regression
from sklearn.linear_model import LogisticRegression
reg=LogisticRegression()  # creating object
reg.fit(X_train,y_train)  # fitting train data using the object
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Figure 5.1.2: Importing,object creation and training

5.1.3 Making Predictions and Testing

1. objectname.predict() is used to predict data
2. It takes input column as parameters
3. Returns numpy arrays

```
#predicting on train data
lg_train_pred=reg.predict(X_train)
#predicting on test data
lg_test_pred=reg.predict(test_inp)
```

Figure 5.1.3: Predicting on training and testing data

5.1.4 Generating Classification Report

```
# Classification report on train and test
from sklearn.metrics import classification_report
print('---------------------------ON TRAIN DATA-------------------------------------')
print(classification_report(y_train,lg_train_pred,digits=4))
print('--------------------------ON TEST DATA-------------------------------------')
print(classification_report(y_test.label,lg_test_pred,digits=4))
```

```
---------------------------ON TRAIN DATA-------------------------------------
              precision    recall  f1-score   support

           0     0.9671    0.9681    0.9676     29720
           1     0.9680    0.9671    0.9676     29720

    accuracy                         0.9676     59440
   macro avg     0.9676    0.9676    0.9676     59440
weighted avg     0.9676    0.9676    0.9676     59440

--------------------------ON TEST DATA-------------------------------------
              precision    recall  f1-score   support

           0     0.9960    0.9393    0.9668     16282
           1     0.4636    0.9333    0.6195       915

    accuracy                         0.9390     17197
   macro avg     0.7298    0.9363    0.7932     17197
weighted avg     0.9677    0.9390    0.9484     17197
```

Figure 5.1.4: Logistic Regression classification report

From the Classification Report ,as we performed SMOTE , accuracy score can be considered as an evaluation metric.

From Classification Report,

- Accuracy score for train data = 0.9676
- Accuracy score for test data = 0.9390

5.1.5 Hyper Parameter Tuning For Logistic Regression

Hyperparameter value will reduce the loss of the model. By specifying a range of possible values for all the hyperparameters. And understand how they are affecting the model performance and architecture.

Grid Search CV: It is a traditional way to perform hyperparameter optimization,it works by searching exhaustively through a specified set of hyperparameters.

Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.

Considering Best Possible Parameters

```
# Taking Parameters for performing HyperParameter Tuning
dual=[True,False]
max_iter= [800]
C = [1.0,1.5,2.0,2.5]
param_grid = dict(dual=dual,max_iter=max_iter,C=C)
```

Figure 5.1.5.1: Setting range of parameters of hyper parameter tuning

Creating new Object and applying GridSearchCV

```
# Creating New Object for Hyper Parameter Tuning
new_lr = LogisticRegression(penalty='l2')
# Importing GridSearchCV for finding Best parameters
from sklearn.model_selection import GridSearchCV
# Initializing Object for GridSearchCV
grid_search = GridSearchCV(estimator=new_lr, param_grid=param_grid, cv = 3, n_jobs=-1)
# Fitting grid_search on Train data
grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=3, error_score=nan,
             estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
                                          fit_intercept=True,
                                          intercept_scaling=1, l1_ratio=None,
                                          max_iter=100, multi_class='auto',
                                          n_jobs=None, penalty='l2',
                                          random_state=None, solver='lbfgs',
                                          tol=0.0001, verbose=0,
                                          warm_start=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [1.0, 1.5, 2.0, 2.5], 'dual': [True, False],
                         'max_iter': [800]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

Figure 5.1.5.2: Performing the GridSearchCV

Printing the best-range of parameters

```
# generating best parameters
grid_search.best_params_
```

```
{'C': 2.5, 'dual': False, 'max_iter': 800}
```

Figure 5.1.5.3: Extracting best parameters

Fitting Best Parameters on Logistic Regression

```
# Creating new object and fitting best parameters
final_lg = LogisticRegression(max_iter = 800, dual=False,C=2.5)
# fitting on train data
final_lg.fit(X_train,y_train)
```

```
LogisticRegression(C=2.5, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=800,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

Figure 5.1.5.4: fitting the new model with best parameters

Predicting on train and test data

```
#predicting on train data
final_lg_train_pred=final_lg.predict(X_train)
#predicting on test data
final_lg_test_pred=final_lg.predict(test_inp)
```

Figure 5.1.5.5: Making predictions on train and test data

Generating Classification Report

```
# After Hyper parameter Tuning
# Classification report on train and test
from sklearn.metrics import classification_report
print('---------------------------ON TRAIN DATA-----------------------------------')
print(classification_report(y_train,final_lg_train_pred,digits=4))
print('-----------------------------ON TEST DATA-----------------------------------')
print(classification_report(y_test.label,final_lg_test_pred,digits=4))
```

```
---------------------------ON TRAIN DATA-----------------------------------
              precision    recall  f1-score   support

           0     0.9889    0.9798    0.9843     29720
           1     0.9800    0.9890    0.9845     29720

    accuracy                         0.9844     59440
   macro avg     0.9844    0.9844    0.9844     59440
weighted avg     0.9844    0.9844    0.9844     59440

-----------------------------ON TEST DATA-----------------------------------
              precision    recall  f1-score   support

           0     0.9941    0.9436    0.9682     16282
           1     0.4727    0.9005    0.6200       915

    accuracy                         0.9413     17197
   macro avg     0.7334    0.9221    0.7941     17197
weighted avg     0.9664    0.9413    0.9497     17197
```

Figure 5.1.5.6: New classification report for Logistic Regression with best parameters

From Classification Report,

- Accuracy score for train data = 0.9844
- Accuracy score for test data = 0.9413

Compared to previous Classification Report before Hyperparameter Tuning

There is a slight increase in accuracy on both train data and test data.

Printing Confusion_Matrix

```
# importing confusion matrix
from sklearn.metrics import confusion_matrix
# printing confusion matrix using heatmap
sns.heatmap(confusion_matrix(y_test.label,final_lg_test_pred),annot = True,fmt = 'd')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fedde2ec6a0>
```

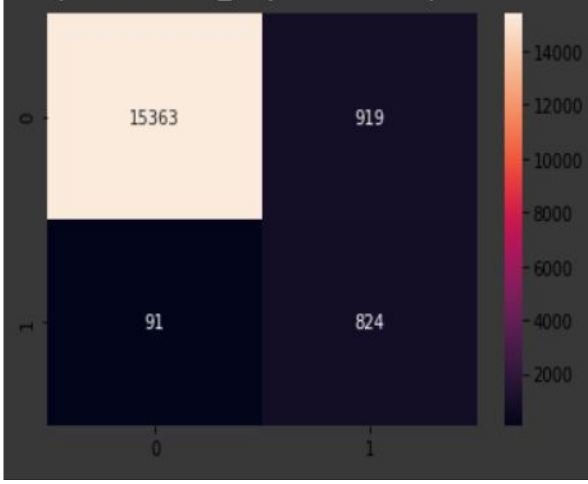


Figure 5.1.5.7 Confusion Matrix

From the Confusion Matrix we get TP = 15363 and TN = 824

5.1.6 Plotting ROC_AUC Curve

ROC(Receiver Operating Characteristic Curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters:

- True Positive Rate
- False Positive Rate

It is a probability curve that plots the TPR against FPR at various threshold values.

The Area Under the Curve(AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

The AUC always ranges from 0 to 1 , such that if AUC = 1, then all True positives are predicted positives and all True Negatives are classified as negative.

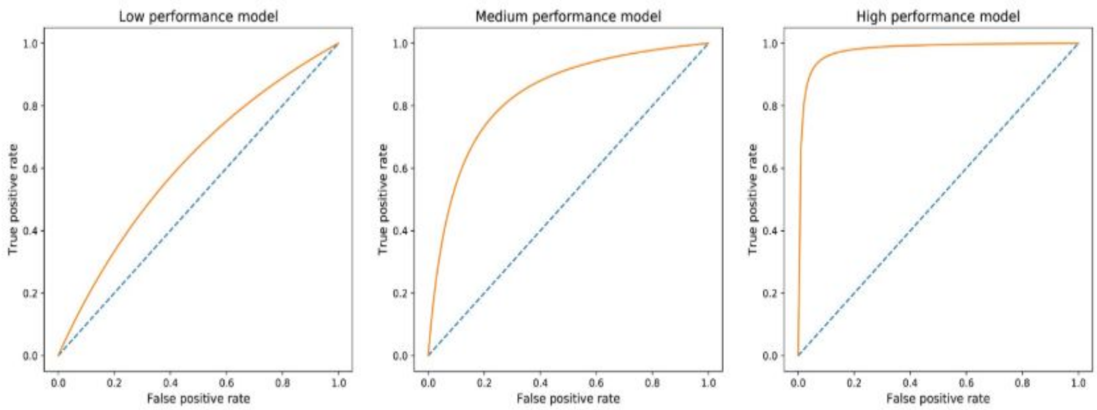


Figure 5.1.6.1 ROC_AUC Curve

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

Creating Objects for ROC_Curve

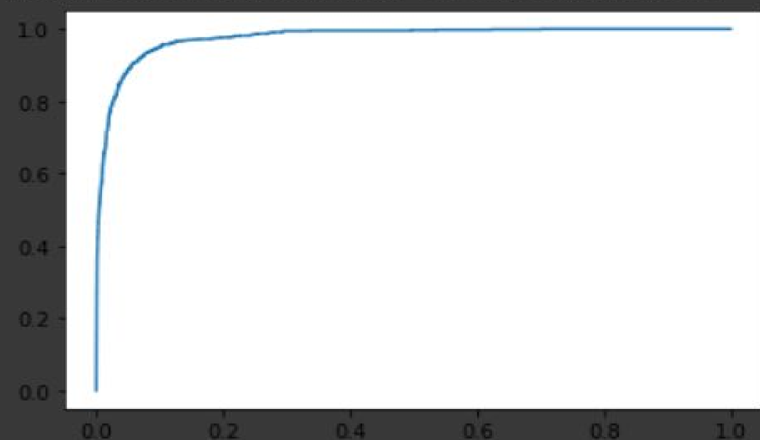
```
# Importing Matplotlib for visualization
import matplotlib.pyplot as plt
# importing roc_auc score and roc_cure
from sklearn.metrics import roc_auc_score,roc_curve
# generating probabilities of any one label
lg_prob = final_lg.predict_proba(test_inp)[:,1]
# creating objects for True positive rates,True Negative rates and threshold
lg_tpr,lg_fpr,lg_threshold = roc_curve(y_test.label,lg_prob)
```

Figure 5.1.6.2 Creating Objects for ROC_Curve

Printing ROC_Curve

```
[94] # Plotting ROC_AUC Curve using tpr and fpr
     plt.plot(lg_tpr,lg_fpr)

[<matplotlib.lines.Line2D at 0x7feddc655d68>]
```



Observations

From the above plot we can say it is High Performance Model

Figure 5.1.6.3 Plotting ROC_Curve

From the Graph we can say that the AUC score is greater than 0.5.Thus, it is a High Performance model.

ROC_Score

```
#ROC Score
roc_auc_score(y_test.label,lg_prob)

0.9768703647394991
```

Figure 5.1.6.4 ROC_Score

5.2 DECISION TREE CLASSIFIER

5.2.1 About Algorithm

Decision tree is one of the predictive modelling approaches used in statistics, data mining and machine learning. Decision trees are constructed via an algorithmic approach that identifies ways to split a data set based on different conditions. It is one of the most widely used and practical methods for supervised learning. Decision Trees are a non-parametric supervised learning method used for both classification and regression tasks.

Decision trees are generated by splitting an attribute as a node. The attribute is selected such that it has higher information gain and lowest entropy.

Entropy:

Entropy is an indicator of how messy your data is.Entropy is the measure of randomness or unpredictability in the dataset. In other terms, it controls how a decision tree decides to split the data. Entropy is the measure of homogeneity in the data. Its value ranges from 0 to 1.It measures the impurity of the split. Gini can also be considered instead of entropy

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

Figure 5.2.1.1: entropy

Information Gain:

Information gain (IG) measures how much "information" a feature gives us about the class.

$$\text{Information gain} = \text{base entropy} - \text{new entropy}$$

$$Information\ Gain = Entropy(before) - \sum_{j=1}^{K} Entropy(j,\ after)$$

Figure 5.2.1.2: information gain

5.2.2 Train the model

We need to import decision tree classifier method from tree package from scikit learn library We need to train the model based on our train set.

1. Importing decision tree classifier
2. Creating Object
3. Fitting on train data

```
# importing decision tree classifier from sklearn.tree package
from sklearn.tree import DecisionTreeClassifier
# creating object
dtree=DecisionTreeClassifier(criterion='entropy')
# Fitting on train data
dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Figure 5.2.2: importing,training and fitting decision tree classifier

5.2.3 Making Predictions and testing

1. objectname.predict() is used to predict data
2. It takes input column as parameters
3. Returns numpy arrays

```
#predicting on train data
dtree_train_pred=dtree.predict(X_train)
#predicting on test data
dtree_test_pred=dtree.predict(test_inp)
```

Figure 5.2.3: predicting on training and testing data

5.2.4 Generating Classification Report

```
# Classification report on train and test
# importing classification report from sklearn.metrics module
from sklearn.metrics import classification_report
print('---------------------------ON TRAIN DATA-----------------------------------')
print(classification_report(y_train,dtree_train_pred,digits=4))
print('---------------------------ON TEST DATA------------------------------------')
print(classification_report(y_test.label,dtree_test_pred,digits=4))
```

```
---------------------------ON TRAIN DATA-----------------------------------
              precision    recall  f1-score   support

           0     0.9999    0.9996    0.9997     29720
           1     0.9996    0.9999    0.9997     29720

    accuracy                         0.9997     59440
   macro avg     0.9997    0.9997    0.9997     59440
weighted avg     0.9997    0.9997    0.9997     59440

---------------------------ON TEST DATA------------------------------------
              precision    recall  f1-score   support

           0     0.9782    0.9451    0.9614     16282
           1     0.3902    0.6251    0.4805       915

    accuracy                         0.9281     17197
   macro avg     0.6842    0.7851    0.7209     17197
weighted avg     0.9469    0.9281    0.9358     17197
```

Figure 5.2.4: Decision tree classifier classification report

From the Classification Report ,as we performed SMOTE , accuracy score can be considered as an evaluation metric.

From Classification Report,

- Accuracy score for train data = 0.9997
- Accuracy score for test data = 0.9281

5.2.5 Hyper Parameter Tuning for the Decision Tree Classifier Model

For Hyper Parameter Tuning Theory refer 6.1.5

Considering Best Possible Parameters

```
# taking best possible parameters
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,32,1),
    'min_samples_leaf' : range(1,10,1)}
```

Figure 5.2.5.1: setting range of parameters for GridSearchCV

Creating new Object and applying GridSearchCV

```
#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV
# initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
clf_new = DecisionTreeClassifier()
grid_search_new= GridSearchCV(estimator=clf_new, param_grid=grid_param,n_jobs=-1,cv=3,verbose=2)
# applying gridsearch onto dataset
grid_search_new.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 540 candidates, totalling 1620 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  37 tasks      | elapsed:   18.3s
[Parallel(n_jobs=-1)]: Done 158 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done 361 tasks      | elapsed:  4.2min
[Parallel(n_jobs=-1)]: Done 644 tasks      | elapsed:  9.9min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 16.2min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 25.3min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 30.0min finished
GridSearchCV(cv=3, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=2)
```

Figure 5.2.5.2: Performing the GridSearchCV

Printing the best-range of parameters

```
grid_search.best_params_
```

```
{'criterion': 'gini', 'max_depth': 31, 'min_samples_leaf': 1}
```

Figure 5.2.5.3: Extracting best parameters

Fitting Best Parameters on Decision Tree Classifier

```
# creating object and applying  best parameters
final_dtree = DecisionTreeClassifier(criterion= 'gini', max_depth= 31, min_samples_leaf= 1)
# fitting object on the model
final_dtree.fit(X_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=31, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

Figure 5.2.5.4: Building new model with best parameters

Predicting on train and test data

```
#predicting on train data
final_dtree_train_pred=final_dtree.predict(X_train)
#predicting on test data
final_dtree_test_pred=final_dtree.predict(test_inp)
```

Figure 5.2.5.5: predicting on training and testing data

Generating Classification Report

```
# Classification report on train and test
from sklearn.metrics import classification_report
print('----------------------ON TRAIN DATA----------------------------------------')
print(classification_report(y_train,final_dtree_train_pred,digits=4))
print('-----------------------ON TEST DATA---------------------------------------')
print(classification_report(y_test.label,final_dtree_test_pred,digits=4))
```

```
----------------------ON TRAIN DATA---------------------------------------
              precision    recall  f1-score   support

           0     0.8558    0.9920    0.9189     29720
           1     0.9905    0.8328    0.9049     29720

    accuracy                         0.9124     59440
   macro avg     0.9232    0.9124    0.9119     59440
weighted avg     0.9232    0.9124    0.9119     59440

-----------------------ON TEST DATA--------------------------------------
              precision    recall  f1-score   support

           0     0.9838    0.9714    0.9776     16282
           1     0.5845    0.7148    0.6431       915

    accuracy                         0.9578     17197
   macro avg     0.7841    0.8431    0.8103     17197
weighted avg     0.9625    0.9578    0.9598     17197
```

Figure 5.2.5.6: New classification report for decision tree classifier with best parameters

From Classification Report,

- Accuracy score for train data = 0.9124
- Accuracy score for test data = 0.9578

Compared to previous Classification Report before Hyperparameter Tuning

There is a slight increase in accuracy on both train data and test data. Overfitting is also resolved by performing Hyperparameter tuning.

Confusion Matrix for Decision Tree Classifier

```
# printing confusion matrix using heatmap
sns.heatmap(confusion_matrix(y_test.label,final_dtree_test_pred),annot = True,fmt = 'd')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feddc7ff208>
```

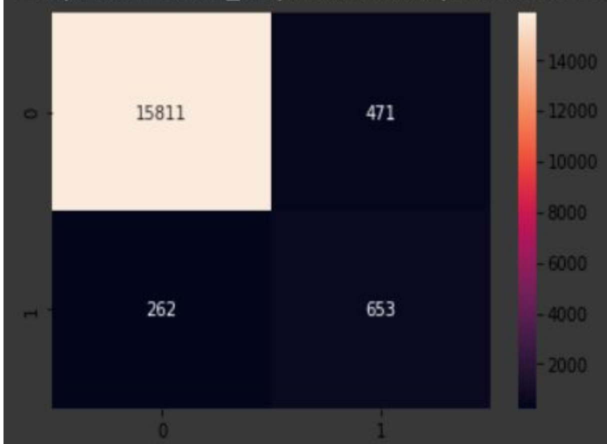


Figure 5.2.5.7 Confusion Matrix

From the Confusion Matrix we get TP = 15811 and TN = 653

5.2.6 ROC_AUC Curve

Creating Objects for ROC_AUC Curve

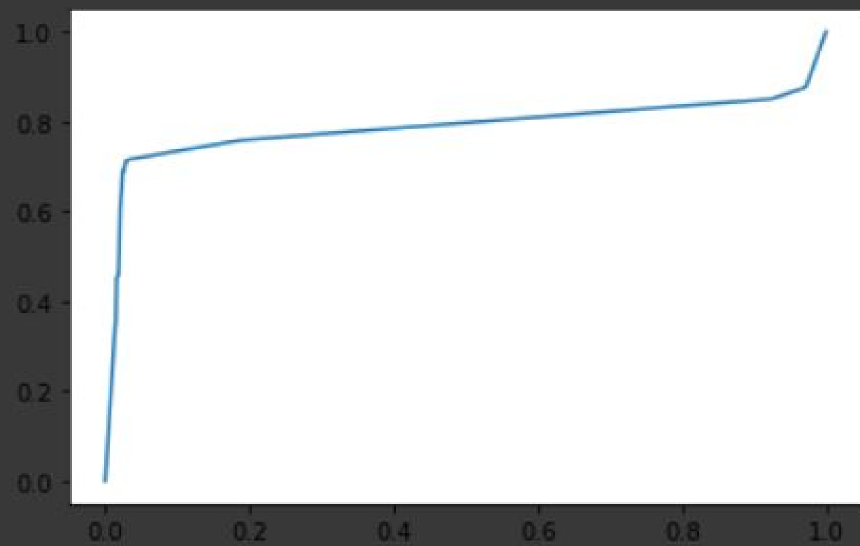
```
# generating probabilities of any one label
dtree_prob = final_dtree.predict_proba(test_inp)[:,1]
# creating objects for True positive rates,True Negative rates and threshold
dtree_tpr,dtree_fpr,dtree_threshold = roc_curve(y_test.label,dtree_prob)
```

Figure 5.2.6.1 Creating Object to plot ROC_AUC Curve

Plotting ROC_AUC Curve

```
[99] # Plotting ROC_AUC Curve using tpr and fpr
     plt.plot(dtree_tpr,dtree_fpr)
```

```
[<matplotlib.lines.Line2D at 0x7feddc53a828>]
```



Observations:

From the plot we can see that it is Low Performance model

Figure 5.2.6.2 Plotting ROC_AUC Curve

From the graph we can see that AUC score is greater than 0.5, but very close to 0.5.

Printing AUC score

```
#ROC Score
roc_auc_score(y_test.label,dtree_prob)

0.7879027965442413
```

Figure 5.2.6.3 printing AUC score

We got an AUC score of 0.7879 , which is only an acceptable score, not a high score.

Choosing Final Algorithm : After applying Logistic Regression and Decision Tree Classifier algorithms, and Performing Hyperparameter Tuning and Plotting ROC_AUC Curve, we observed that the Logistic Regression algorithm has given more accuracy and also AUC score is much better than Decision Tree Classifier.

So, Using Logistic Regression algorithm for prediction on new data.

CHAPTER 6

SCRAPING REAL-TIME TWITTER DATA

6.1 GET OLD TWEETS 3 LIBRARY:

6.1.1 About the library

- A project written in Python to get old tweets, it bypasses some limitations of Twitter Official API.

  Advantages:

    - It does not require a twitter developer account as the data used is public
    - Unlike the twitter API ,tweets more than one week old can be extracted.
    - The coding is relatively easy.

6.1.2 Python classes

Tweet: Model class that describes a specific tweet.

- id (str)
- username (str)
- to (str)
- text (str)
- date (datetime) in UTC

TweetManager: A manager class to help getting tweets in Tweet's model.

- getTweets (TwitterCriteria): Return the list of tweets retrieved by using an instance of TwitterCriteria.

TwitterCriteria: A collection of search parameters to be used together with TweetManager.

- setUsername (str or iterable): An optional specific username(s) from a twitter account (with or without "@").
- setSince (str. "yyyy-mm-dd"): A lower bound date (UTC) to restrict search.
- setUntil (str. "yyyy-mm-dd"): An upper bound date not included to restrict search.
- setQuerySearch (str): A query text to be matched.
- setMaxTweets (int): The maximum number of tweets to be retrieved. If this number is unsettled or lower than 1 all possible tweets will be retrieved.

6.1.3 Applying the GetOldTweets:

```
[ ] import GetOldTweets3 as got
    tag=input("Enter the topic to run sentiment analysis on :")
    limit=300
```

Figure 6.1.3: import GetOldTweets, read input for 'blacklivesmatter'

6.1.4 Initializing TweetCriteria and getting tweets

```
# setting search criteria using TweetCriteria model
# setQuerySearch takes the input "tag" previously given by user
# setMaxTweets takes number of tweets to generate as parameter
tweetCriteria = got.manager.TweetCriteria().setQuerySearch(tag)\
                                            .setMaxTweets(limit)
# getting tweets from TweetCriteria
tweet = got.manager.TweetManager.getTweets(tweetCriteria)
```

Figure 6.1.4: setting criteria and extracting tweets

6.1.5 Converting to DataFrame

```
# tweets contain 2 columns time and tweet
# formatting into DataFrame
tweet_in = [[i.date, i.text] for i in tweet]
input = pd.DataFrame(tweet_in)
input.head() # new DataFrame
```

	0	1
0	2020-07-13 16:59:30+00:00	cade os black lives matter falando do policial...
1	2020-07-13 16:59:30+00:00	#BlackLivesMatter #StephonClark
2	2020-07-13 16:59:29+00:00	Allowing black and ethnic students to share ex...
3	2020-07-13 16:59:26+00:00	@Dbongino, All #BlackLivesMatter supporters sh...
4	2020-07-13 16:59:26+00:00	#BlackLivesMatter #Clinton #SusanRosenberg #BL...

Figure 6.1.5: formatting to data frame

6.1.6 Predicting on input data

```
# Predicting Input data
k = final_lg.predict(tfidf.transform(input.iloc[:,0]))
```

Figure 6.1.6: predicting on unseen data

6.1.7 Adding predicted data to the input data frame

```
# adding predicted labels to input data
input['label'] = pd.DataFrame(k)
```

```
# Visualizing newly created column
input.head()
```

	1	label
0	I like to see Young black men on their grind d...	1
1	IMF Executive Board Approves US$4.3 Billion in...	1
2	See that's what Republicans and The klansman d...	0
3	To obadaj to https://youtu.be/DGk6kULEJtU #Bla...	1
4	All these punk ass rappers making black lives ...	1

Figure 6.1.7: formatting to data frame and appending to column

6.1.8 Checking Output Column Values

```
# Checking value_counts
input.label.value_counts()
```

```
1    242
0     58
Name: label, dtype: int64
```

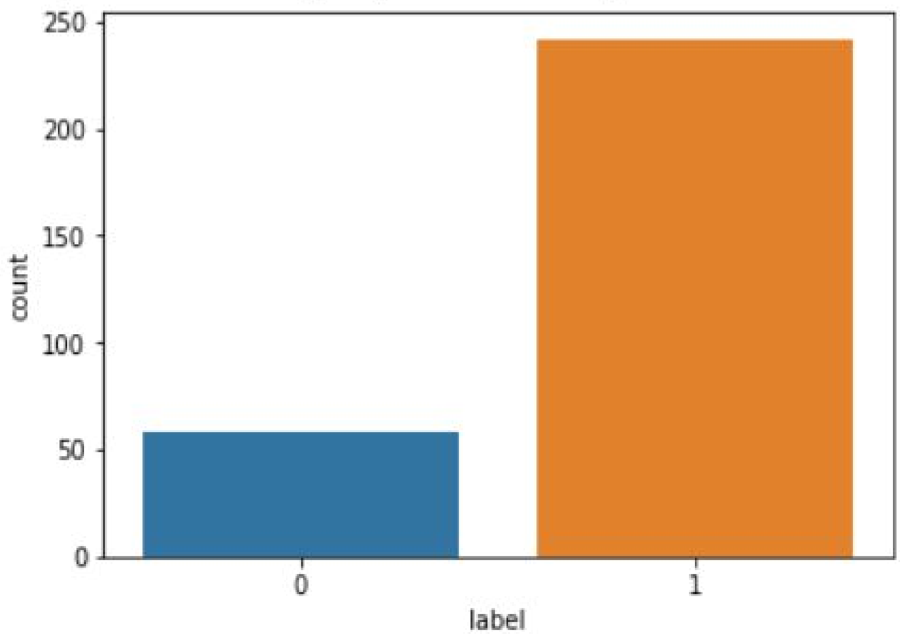
Figure 6.1.8: value counts of unseen data

6.1.9Visualizing On Predicted Data

Using CountPlot

```
# Visualizing predicted column
sns.countplot(input.label)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7feddc3aa748>
```



Observations:

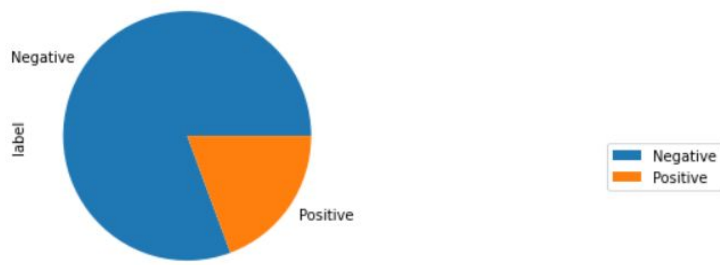
```
'0' represents Positve
'1' represents Negative
The "blacklivesmatter" has mostly negative sentiment in twitter
```

Figure 6.1.9.1: Countplot of positive and negative tweets

Using Pie Plot

```
[108] # Plotting on pie chart
      %matplotlib inline
      input.label.value_counts().plot.pie(labels=['Negative','Positive']).legend(labels=['Negative','Positive'],
                                                  bbox_to_anchor=(2.25,0.5))
```

```
<matplotlib.legend.Legend at 0x7feddc3ed6d8>
```



Observations:

```
Negative tweets are more in number than positive tweets
```

Figure 6.1.9.2: Pie chart for the distribution of tweets

CONCLUSION

In This Project we tried to classify the tweets into positive or negative categories using Logistic Regression and Decision Tree Classifier algorithms.Firstly, we tried to clean the data using nltk and handling imbalance dataset by SMOTE. Then we compared both the algorithms and generated the classification report which has shown unsatisfying figures,then we performed hyper parameter tuning for generating the best range of parameters and ROC_AUC score for both algorithms , from which we understood that Logistic Regression has shown much better results than Decision Tree Classifier.

So , we used Logistic Regression for Unseen data using getoldtweets library and predicted recent trending tweets on '#blacklivesmatter', whose results showed that there are more negative tweets than positive tweets.

References:

GITHUB LINK : https://github.com/ShoaibShaik786/Project_Twitter_Sentiment_Analysis

1.https://stackoverflow.com/questions/8376691/how-to-remove-hashtag-user-link-of-a-tweet-using-regular-expression

2.https://medium.com/@SeoJaeDuk/basic-data-cleaning-engineering-session-twitter-sentiment-data-b9376a91109b

3.https://stackoverflow.com/questions/29523254/python-remove-stop-words-from-pandas-dataframe

4.https://stackoverflow.com/questions/771918/how-do-i-do-word-stemming-or-lemmatization

5.https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

6.https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

7.https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

8.https://pypi.org/project/GetOldTweets3/