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

Vishnu Pulipaka

221710302064

Under the Guidance of

Department Of Computer Science Engineering

GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

July 2020

i
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.
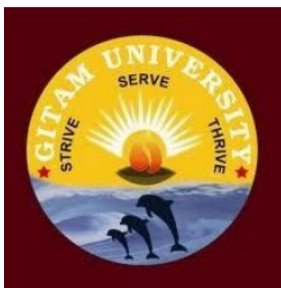
Place: HYDERABAD   Vishnu Pulipaka

Date :14th July,2020   221710302064

ii

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 **Vishnu Pulipaka
(221710302064)** 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,
GITAM University Hyderabad Campus under my guidance and supervision.

Kumar   **Dr.S.Phani**

Assistant Professor   Professor and HOD

Department of CSE   Department of CSE

iii

iv
ACKNOWLEDGEMENT

Vishnu Pulipaka

221710302064

v
ABSTRACT

Classification algorithms in supervised machine learning are used to predict the label of a given "tweet" as positive or negative based on it's content. The text data is converted from the perception of a mode of expression to semantically equivalent numeric format so as to be used for training,testing and as input for the machine learning model developed.The client's requirement of understanding sentiment among twitter users who may be consumers, viewers or any service receivers from the client's entity is the key objective of this case study.

To understand the necessity of sentiment analysis, I have put myself in the perspective of the client aiming to explore the scope of advertisement of my services and consumer mentality pertaining to a product,service,business or a social movement. In general, this case study can be used to predict sentiment among buyers,commuters and even sports fans.etc. by searching the relevant topics. There is a scope of expansion to study the variation of sentiment with time,location and user etc.

vi
Table of Contents:

viii

ix
LIST OF FIGURES:

x

xi

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
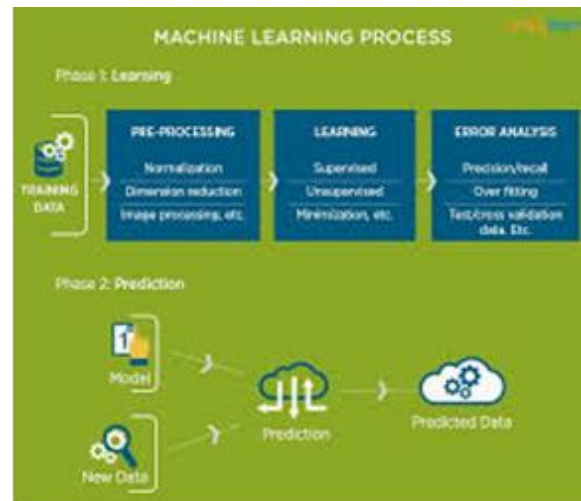
Figure 1 : The Process Flow

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.
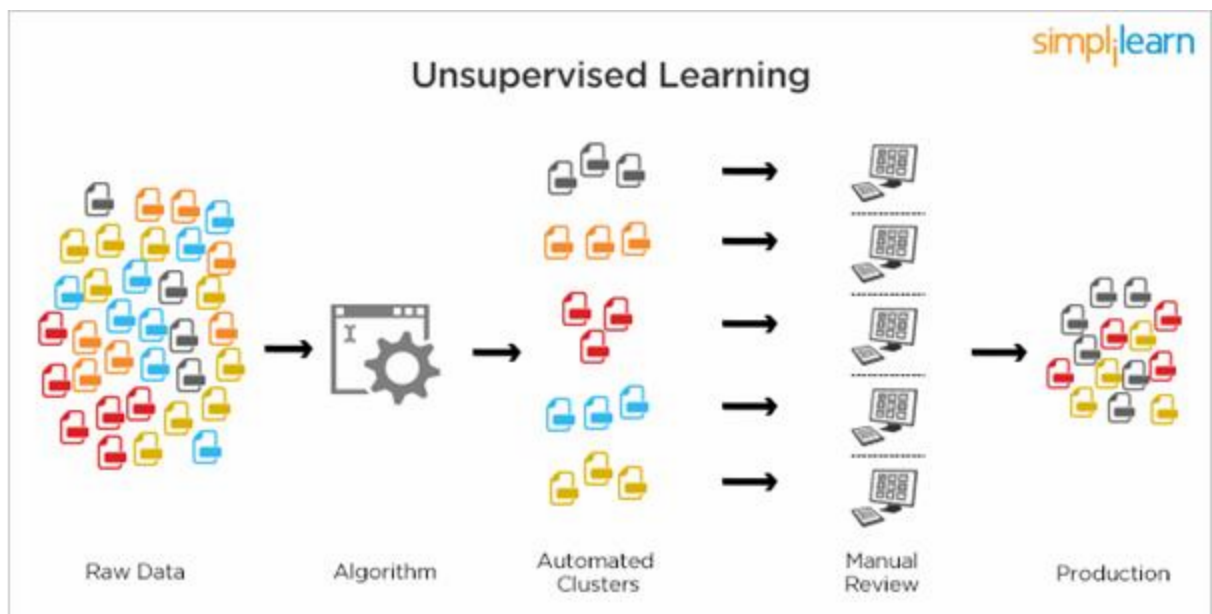
Figure 2 : Unsupervised Learning

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.
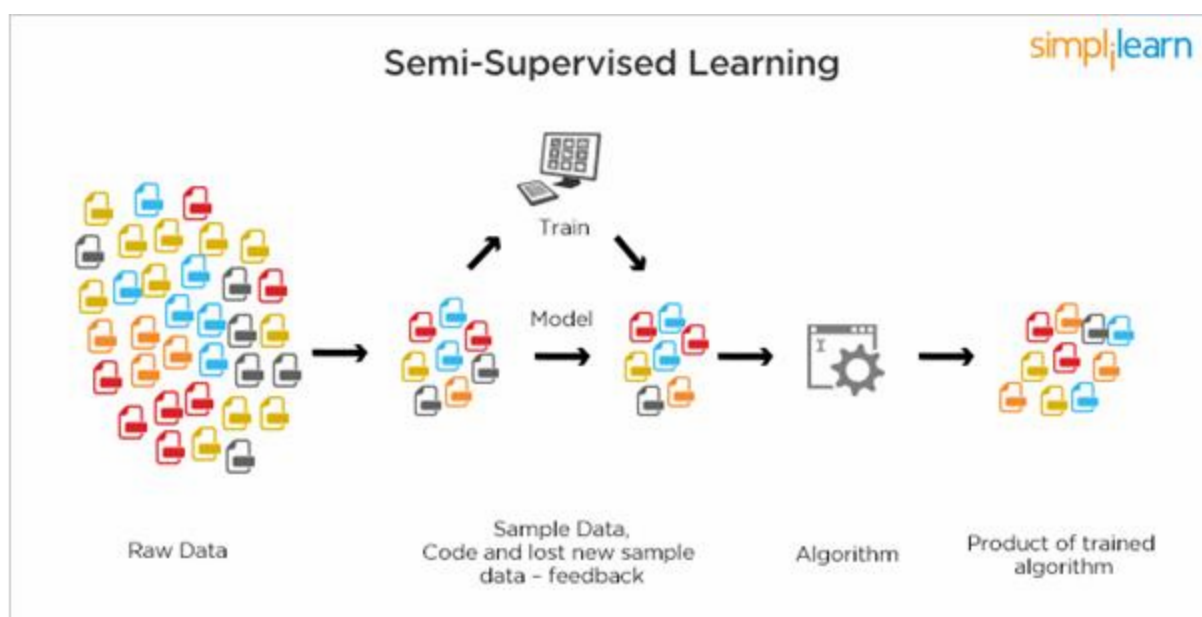
Figure 3 : Semi Supervised Learning

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 discovers 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 4 : 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 quickly installs and manages packages.

In WINDOWS:

- In windows

- Step 1: Open Anaconda.com/downloads in web browser.

- Step 2: Download python 3.4 version for (32-bitgraphic 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)
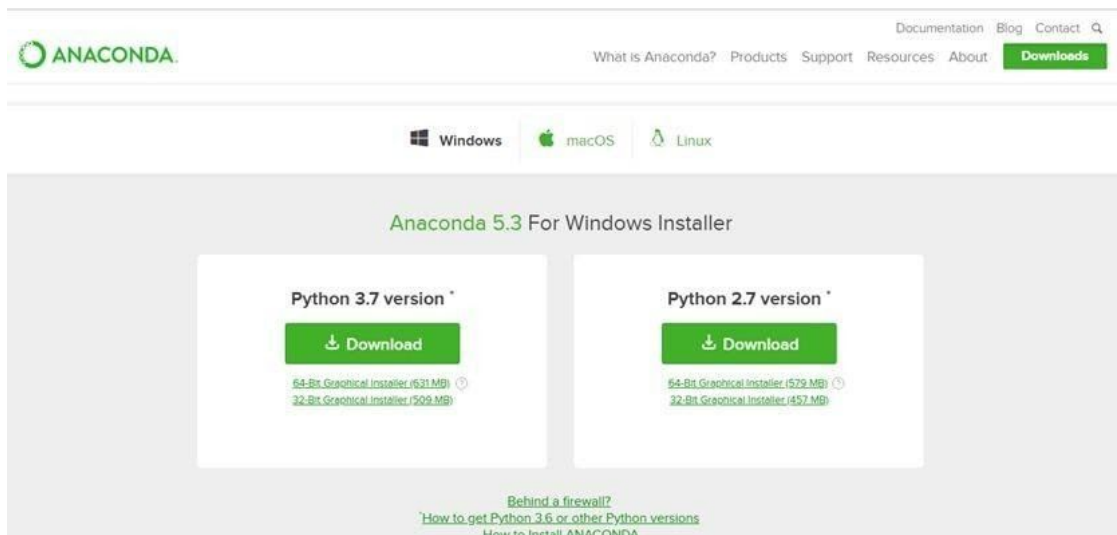
Figure 5 : Anaconda download

Figure 6 : 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 –

    o   Numbers

    o   Strings

    o   Lists

    o   Tuples

    o   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 7 : 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
CASE STUDY: TWITTER SENTIMENT ANALYSIS

3.1 PROBLEM STATEMENT:

" To predict the sentiment of twitter data using machine learning algorithms - Decision Tree Classifier and Logistic Regression".

3.2 DATA SET:

ID - tweet ID

Label - 0/1 (positive/negative)

Tweet - text data(tweets)

3.3 OBJECTIVE OF THE CASE STUDY:

Analysing sentiment of client's domain related topic from twitter data using machine learning algorithms. Primary objective is to find the best suitable algorithm for the implementation of sentiment analysis and tune parameters for optimum accuracy.

Further helping the client to learn about the response of the consumer/public about a topic pertaining to their business domain.

CHAPTER 4

DATA PREPROCESSING

4.1 READING DATA AND LIBRARIES:

Preparing text using following steps:

4.1.1 Getting the dataset:

We can get the data set from twitter by scraping the web.

4.1.2 Importing the libraries:

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

Figure 8: Importing libraries

4.1.3 Importing the data :

We read the training data and printing the head.

```
[ ]  # Reading the dataset
     tweets=pd.read_csv("/content/drive/My Drive/Datasets/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 9: Reading the data and head of the data

4.2 STATISTICAL ANALYSIS :

4.2.1 Checking the frequency of output data:

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

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

Figure 10: label counts

4.2.2 Visualising Output Column:

```
[ ]  sns.countplot(tweets.label)
```

```
 ⤷   <matplotlib.axes._subplots.AxesSubplot at 0x7f622e12d7f0>
```
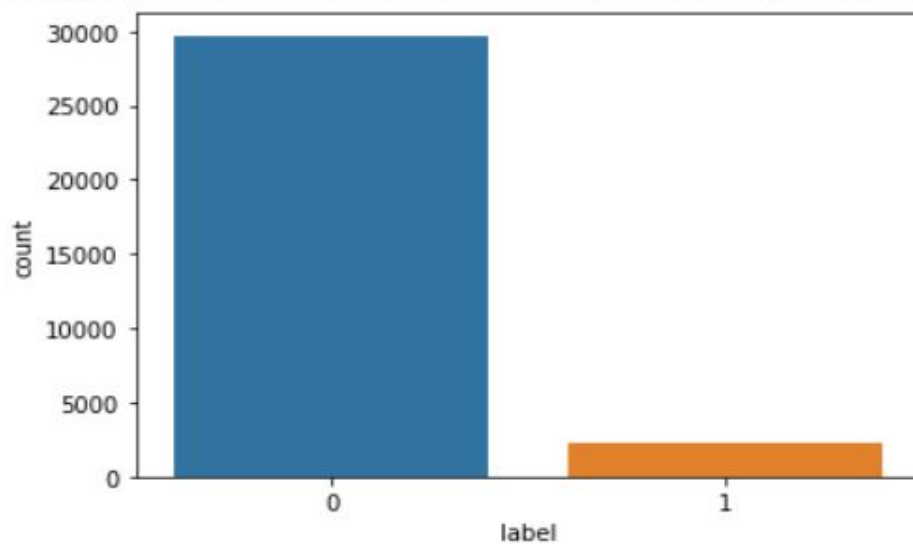

Figure 11:visualizing label counts

- **Observation** : There is an imbalance in the data as label '0' and label '1' data are approximately in the ratio 15:1 . The data is imbalanced. Data is balanced in the upcoming section 5.5.

4.2.3 Checking Shape Of Data Set:
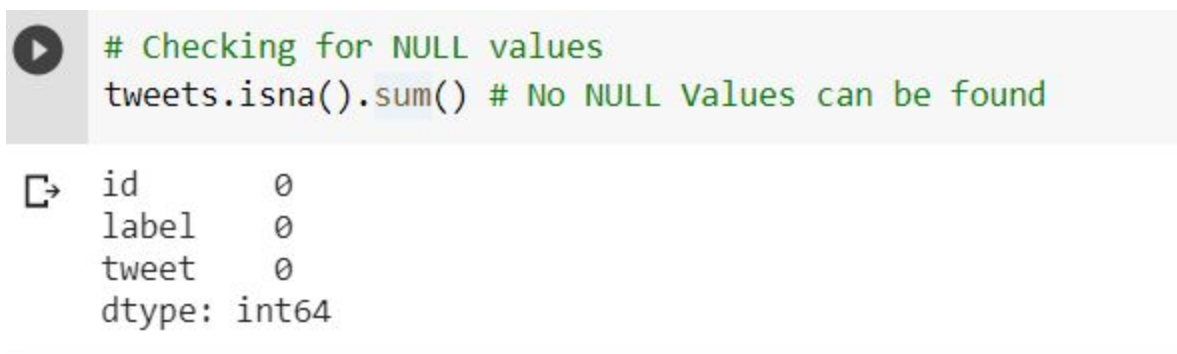
Figure 12: Shape of dataset

4.2.4 Checking Statistical Description:

	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 13: 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 14: Checking missing values

- No missing values found in the dataset .

- Hence, no need for imputation .

4.4 CLEANING TEXT DATA WITH NLTK:

4.4.1 Downloading Resources And Nltk Library:

- List of stopwords

```
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 15: nltk stopwords download

- WordNet

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

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

Figure 16: nltk wordnet download

4.4.2 Removing Stop Words:

Stopwords:

stopwords are words like [a, an ,the ,is ,are...etc] that are not useful for the prediction.

```
# Removing Stopwords
# stopwords are words like [a, an ,the ,is ,are...etc] that are not useful for the prediction
from nltk.corpus import stopwords
stop=stopwords.words("english")
stop.extend(["i'm","I'm"])

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 17: Removing stopwords

Example : "I am having a bad day" becomes " having bad day"

- am,I,a are removed as they are the stopwords

4.4.3 Removing Hyperlinks,Mentions And Special Characters:

```python
# a function to clear the text data from user-mentions , special characters, hyperlinks
# using regular expressions
def clean(x):
    x=' '.join(re.sub("(@[A-Za-z0-9]+)|([^A-Za-z0-9'])+)|(\w+:\/\/\S+)"," ",x).split())
    return x
```

```python
# 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 18: Removing Hyperlinks

For example:

(@[A-Za-z0-9]+) -   @peter   removed

([^A-Za-z0-9']+) -   #Macy   Macy

(\w+:\/\/\S+)   http://bit.ly//WjdiW4   removed

"@peter I really don't love that shirt at #Macy. http://bit.ly//WjdiW4" becomes ,

"I really don't love that shirt at Macy"

4.4.4 Lemmatization:

```
[ ]  # 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 19: Lemmatization

Example:

"Playing" is converted to the root word "Play".

4.4.5 Converting To Lowercase:

- Converting all words to lowercase for convenience.

```
tweets.tweet=tweets.tweet.apply(lambda x:' '.join([word.lower() for word in x.split()]))
```

Figure 20: Lowercase conversion

4.4.6 Applying The Above Techniques To Test Data

```
[ ]  #reading test data input into X_test
     X_test = pd.read_csv('/content/drive/My Drive/Datasets/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 21: Applying all cleaning techniques to test data

```
# reading the test output into y_test
y_test = pd.read_csv('/content/drive/My Drive/Datasets/result.csv')
```

Figure 22: Reading test data output

4.5 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()
```

```
#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
```

Figure 23: Tfidf Vectorizer

4.6 Balancing Dataset:

Imbalanced data typically refers to a classification problem where the number of observations per class is not equally distributed.

Since most Machine Learning algorithms are designed to achieve maximum accuracy, they tend to classify only the class with higher data population to achieve this accuracy. Since this accuracy is only biased, the accuracy cannot be considered as genuine.

As we have seen in sections 5.2.1 and 5.2.2, the data is imbalanced which may lead to falsely high metrics. To avoid this we use data balancing techniques. The technique used in this case study is Synthetic sample generation.

It is a technique similar to upsampling where the sample count of the minority class is increased. Instead of repeated duplication of samples, here we use imblearn's SMOTE or Synthetic Minority Oversampling Technique. SMOTE uses a nearest neighbours algorithm to generate new and synthetic data we can use for training out model.
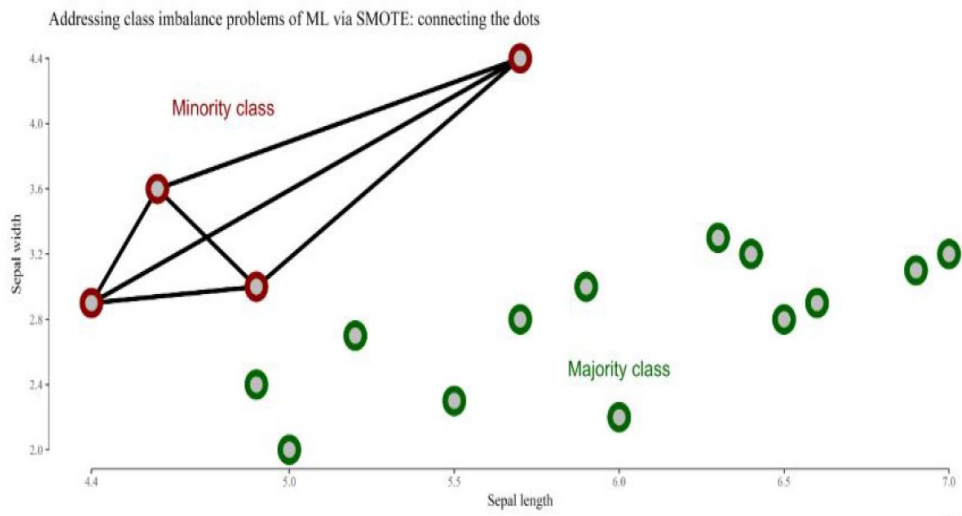
Figure 24: SMOTE Lines

In the above image, the minority classes represented in red, are clearly outnumbered. The SMOTE technique assumes more minority samples plotted on the SMOTE lines used to join the minority data points as seen below.
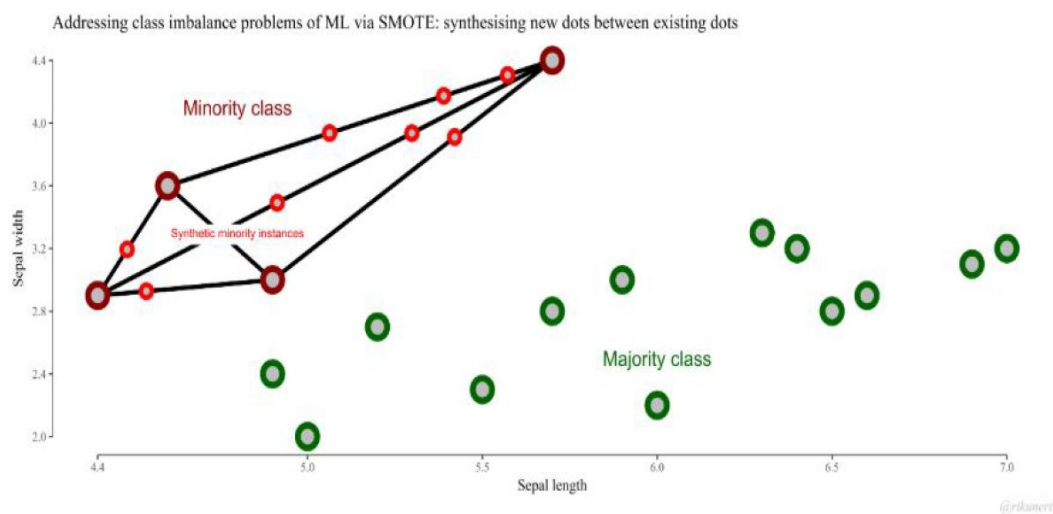
Figure 25: Synthetic data point for minority class

Applying SMOTE on our dataset:

```
[20] # importing the SMOTE tool
     from imblearn.combine import SMOTETomek

     # creating an object
     smk = SMOTETomek(random_state=42)

     # generating the input and output training data after applying the sythentic generation
     X_train,y_train=smk.fit_sample(new_inp,tweets.label)
```

Figure 26: Applying smote to data

1. Importing the SMOTE library and SMOTETomek
2. Creating the object
3. Generating the train data - input(X_train) and output(y_train)

Checking if the synthetic generation has worked:

```
check=pd.DataFrame(y_train)          # converting y_train from np.array to data frame to check for the balancing outcome
check.iloc[:,0].value_counts()       #  using index to generate counts of the label

## we can see that both the labels are no equalized

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

Figure 27: value counts for label

As we can see, both the labels (0/1) have the same frequency as compared to the original data shown below.

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

     ## the label counts of the original data before synthetic generation.

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

Figure 28: value counts before SMOTE

Final Outcome of the step:

The frequency count of the minority label(1) is sampled up from 2242 to 29720 samples by adding synthetically generated data ,in this case using K-means clustering algorithm.

CHAPTER 5

BUILDING MODEL AND EVALUATION

5.1 LOGISTIC REGRESSION

5.1.1 About The Algorithm:

Logistic Regression is used when the dependent variable(target) is categorical.
Logistic Regression is the appropriate regression analysis to conduct when the dependent variable is binary. Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability.
Also called a classification algorithm, because it classifies the data. It predicts the probability associated with each dependent variable category.

- The algorithm finds a linear relationship between independent variables and a link function of these probabilities which provides the best of goodness fit for the given data is chosen.
- The below formula is used to keep the probability in between 0 and 1.

The probability in a logistic regression curve

$$p = \frac{e^y}{1 + e^y}$$

Where,

e is a real number constant, the base of natural logarithm and equals 2.7183

y is the response value for an observation

Figure 29: Probability function for logistic regression

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)

- Then we need to import the Logistic regression method from the linear_model package from scikit learn library.

- We need to train the model based on our train set.
  - Import the model
  - Creating an object
  - Fitting the object on training data.

```
# importing the model
from sklearn.linear_model import LogisticRegression

# creating object
reg=LogisticRegression()

# fitting object on training data
reg.fit(X_train,y_train)
```

```
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 30: Importing,object creation and training

● Now we have a trained Logistic Regression model "reg".

5.1.3 Making Predictions And Testing :

- Predict method of the object is used to make the predictions.
- It takes input data as an argument and returns a numpy array containing the predictions.

Predicting on training and testing data.

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

Figure 31: Predicting on training and testing data

- We have the predicted output on both training and testing data.

5.1.4 Generating Classification Report :

● We generate the classification report for both training and testing data.

```
# Classification report on train and test
from sklearn.metrics import classification_report,recall_score
print("on training data :")
print(classification_report(y_train,lg_train_pred,digits=4))
print('----------------------------------------------------------------')
print('on testing data :')
print(classification_report(y_test.label,lg_test_pred,digits=4))
```

```
on training 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 testing 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 32: Logistic Regression classification report

● Since we perform synthetic sample generation to balance the data, we can consider the accuracy score as an evaluation metric.
● We observe that the accuracy score for :

Training data : 0.9676

Testing data : 0.9390

Which implies correct predictions are made for 93.9 % of the time on test data.

5.1.5 Hyperparameter Tuning For Logistic Regression:

- A hyperparameter is a parameter whose value is set before the learning process begins.
- Hyperparameter value will reduce the loss of the model,hence,increasing the evaluation metric
- Hyperparameters are taken by specifying the range of possible values for the parameters of that model and understand how they are affecting the model architecture and performance.

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 the range of 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 33: Setting range of parameters of hyper parameter tuning

Performing the 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 34: Performing the GridSearchCV

Generating best parameters:

```
grid_search.best_params_

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

Figure 35: Extracting best parameters

Creating a new model and fitting best parameters:

```
| # creating a model with generated best parameters
  new_lr=LogisticRegression(C=2.5,dual=False,max_iter=800)

  # fitting on training data:
  new_lr.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 36: fitting the new model with best parameters

Predicting training and testing data:

```
[ ]  # prediciton on training data
     y_train_pred_tuned_LR=new_lr.predict(X_train)

     # predicition on testing data
     y_test_pred_tuned_LR=new_lr.predict(test_inp)
```

Figure 37: Making predictions on train and test data

New classification report:

```
# Classification report on train and test
from sklearn.metrics import classification_report
print("on training data: ")
print(classification_report(y_train,y_train_pred_tuned_LR,digits=4))
print("on testing data: ")
print('------------------------------------------------------------')
print(classification_report(y_test.label,y_test_pred_tuned_LR,digits=4))
```

```
on training 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 testing 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 38: New classification report for Logistic Regression with best parameters

Observation:

The accuracy increases from 0.9390 to 0.9413 after hyper parameter tuning.

5.2 DECISION TREE CLASSIFIER:

5.2.1 About The 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 to be split is selected such that it has highest 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 index is also used instead of entropy.

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

Figure 39: entropy

Information gain:

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

Information gain = base entropy — new entropy

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

Figure 40: information gain

5.2.2 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)
- Then we need to import the Decision tree classifier method from the tree package from scikit learn library.
- We need to train the model based on our train set.
  1. Import the model
  2. Creating an object
  3. Fitting the object on training data.

```
[68]  # import the library
      from sklearn.tree import DecisionTreeClassifier
      # create object
      dtree=DecisionTreeClassifier(criterion='entropy')
      # fitting the 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 41: importing,training and fitting decision tree classifier

5.2.3 Making Predictions And Testing :

- Predict method of the object is used to make the predictions.
- It takes input data as an argument and returns a numpy array containing the predictions.

Predicting on training and testing data.

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

Figure 42: predicting on training and testing data

- Now we have the predicted output for training and testing data

5.2.4 Generating Classification Report :

- We generate the classification report for both training and testing data.

```
# Classification report on train and test
from sklearn.metrics import classification_report
print("on training data")
print(classification_report(y_train,dtree_train_pred,digits=4))
print("on testing data")
print('------------------------------------------------------------')
print(classification_report(y_test.label,dtree_test_pred,digits=4))
```

```
on training 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 testing data
------------------------------------------------------------
              precision    recall  f1-score   support

           0     0.9778    0.9447    0.9610     16282
           1     0.3858    0.6186    0.4752       915

    accuracy                         0.9273     17197
   macro avg     0.6818    0.7816    0.7181     17197
weighted avg     0.9463    0.9273    0.9351     17197
```

Figure 43: Decision tree classifier classification report

Since we perform synthetic sample generation to balance the data, we can consider the accuracy score as an evaluation metric.

- We observe that the accuracy score for :
- Training data : 0.9991
- Testing data : 0.9273

Which implies correct predictions are made for 92.7 % of the time on test data.

5.2.5 Hyperparameter Tuning For Decision Tree Classifier:

Hyperparameter tuning explained in section 6.1.5

Considering the range of parameters:

```
[72]  #  setting range of parameters
      grid_param = {
           'criterion': ['gini', 'entropy'],
           'max_depth' : range(2,32,1),
           'min_samples_leaf' : range(1,10,1)

      }
```

Figure 44: setting range of parameters for GridSearchCV

Performing the GridSearchCV:

```
#Import the GridSearchCV
from sklearn.model_selection import GridSearchCV

# initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
clf = DecisionTreeClassifier()
grid_search = GridSearchCV(estimator=clf, param_grid=grid_param,n_jobs=-1,cv=3,verbose=2)

# applying gridsearch onto dataset
grid_search.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:   18.9s
[Parallel(n_jobs=-1)]: Done  158 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done  361 tasks      | elapsed:  4.4min
[Parallel(n_jobs=-1)]: Done  644 tasks      | elapsed: 10.6min
[Parallel(n_jobs=-1)]: Done 1009 tasks      | elapsed: 17.5min
[Parallel(n_jobs=-1)]: Done 1454 tasks      | elapsed: 27.4min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 32.5min 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 45: Performing the GridSearchCV

Generating best parameters:

```
# generating best params
grid_search.best_params_
```

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

Figure 46: Extracting best parameters

Creating a new model and fitting best parameters:

```
# creating model with best parameters:
clf=DecisionTreeClassifier(criterion='gini',max_depth=31,min_samples_leaf=1)
```

Figure 47: Building new model with best parameters

```
clf.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 48: fitting the new model

Predicting training and testing data:

```
[75]  ## predicting on training:
      y_train_pred_tuned=clf.predict(X_train)
      ## predicting on testing data:
      y_test_pred_tuned=clf.predict(test_inp)
```

Figure 49: predicting on training and testing data

New classification report:

```
[76] # Classification report on train and test after hyper tuning for decision tree
     from sklearn.metrics import classification_report
     print(classification_report(y_train,y_train_pred_tuned,digits=4))
     print('----------------------------------------------------------------')
     print(classification_report(y_test.label,y_test_pred_tuned,digits=4))
```

```
                 precision    recall  f1-score   support

             0     0.8559    0.9921    0.9190     29720
             1     0.9906    0.8329    0.9050     29720

      accuracy                         0.9125     59440
     macro avg     0.9232    0.9125    0.9120     59440
  weighted avg     0.9232    0.9125    0.9120     59440


     ---------------------------------------------------------------
                 precision    recall  f1-score   support

             0     0.9839    0.9714    0.9776     16282
             1     0.5852    0.7169    0.6444       915

      accuracy                         0.9579     17197
     macro avg     0.7845    0.8442    0.8110     17197
  weighted avg     0.9627    0.9579    0.9599     17197
```

Figure 50: New classification report for decision tree classifier with best parameters

Observations:

- Overfitting is balanced as train accuracy comes closer to test accuracy .
- The accuracy on the test data has increased from 0.92 to 0.9579.

OUTCOME OF THIS STEP:

After building,testing and evaluation of both models ,Linear Regression and decision tree classifier, it is concluded that the decision tree classifier is a best suitable algorithm with

higher efficiency on test data and resolved overfitting through hyper parameter tuning. Hence,this model is used in the following section to predict and visualize real-time 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 bypass 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 that 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 the library :

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

Enter the topic to run sentiment analysis on :feminist movement

Figure 51: import GetOldTweets, read input for topic

"Feminist movement" is the topic entered by the user.

Setting criteria and extracting tweets:

```
[ ]  # setting search criteria using tweetCriteria model
     tweetCriteria = got.manager.TweetCriteria().setQuerySearch(tag)\          # tag from user
                                          .setMaxTweets(limit)                 # limit of maximum tweets to search

     # get tweets using getTweets method:
     tweet = got.manager.TweetManager.getTweets(tweetCriteria)
```

Figure 52: setting criteria and extracting tweets

Formatting to dataframe:

```
# converting the "data" and "text" attributes from model to list of lists:
tweet_in = [[i.date, i.text] for i in tweet]
```

Figure 53: formatting to data frame

```
[ ]  # converting the list of lists to dataframe:
     tweet_in=pd.DataFrame(tweet_in,columns=["time","text"])
     tweet_in.head()
```

	time	text
0	2020-07-13 13:23:33+00:00	Most men are amazing Most women are amazing Do...
1	2020-07-13 12:33:08+00:00	What makes me even more complicated is the fac...
2	2020-07-13 12:04:28+00:00	Well, masculinity is the main virtue praised i...
3	2020-07-13 11:56:58+00:00	Men have been consistently undermined and deva...
4	2020-07-13 11:37:03+00:00	Please join this live discussion via Y-FEM the...

Figure 54: formatting to data frame

Making predictions:

```
#making the predictions by converting the dataFrame by using tfidf vectorizer
real_world_pred=clf.predict(tfidf.transform(tweet_in.iloc[:,1]))
```

Figure 55: predicting on new unseen data

```
# Creating the output column
tweet_in['label']=real_world_pred
```

Figure 56: appending output column

Final Predictions:

```
[130] tweet_in.head()
```

	time	text	label
0	2020-07-13 16:43:33+00:00	As a feminist, I personally don't believe any ...	1
1	2020-07-13 16:39:01+00:00	You go onto the centrist or right to let em kn...	1
2	2020-07-13 16:33:04+00:00	Important to classify feminism as a political ...	0
3	2020-07-13 16:31:16+00:00	And no, it's not all women or feminists, obvio...	1
4	2020-07-13 16:06:15+00:00	You do realize the lesbian movement partnered ...	1

Figure 57: Final Predictions

6.1.4 Visualizing the predictions:

Using countplot:

```
[147] import matplotlib.pyplot as plt
      plt.xlabel("Sentiment label")
      plt.ylabel("count")
      sns.countplot(real_world_pred)
      # 0- positive tweets
      # 1- negative tweets
      plt.show()
```
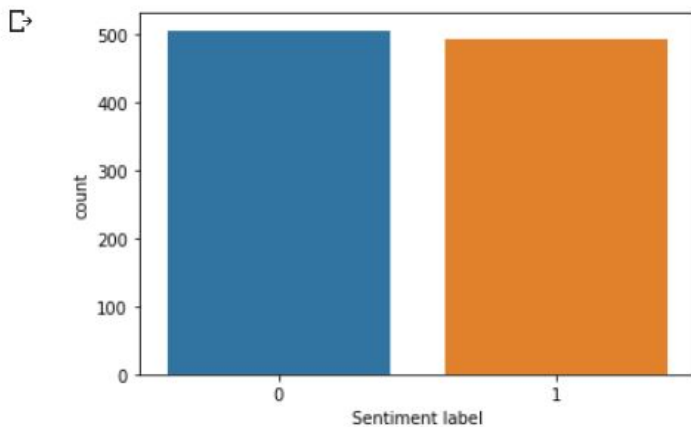

Observations :

- The topic "feminist movement" seems to have almost equally divided sentiment over twitter

Figure 58: Countplot of positive and negative tweets

Observations:

The topic "feminist movement" seems to have almost equally divided sentiment over twitter.

Using line plot to see fluctuation:

```
[149] plt.figure(figsize=(30,8))
      sns.lineplot(x='time',y='label',data=tweet_in.iloc[:,:10])
      plt.show()
```
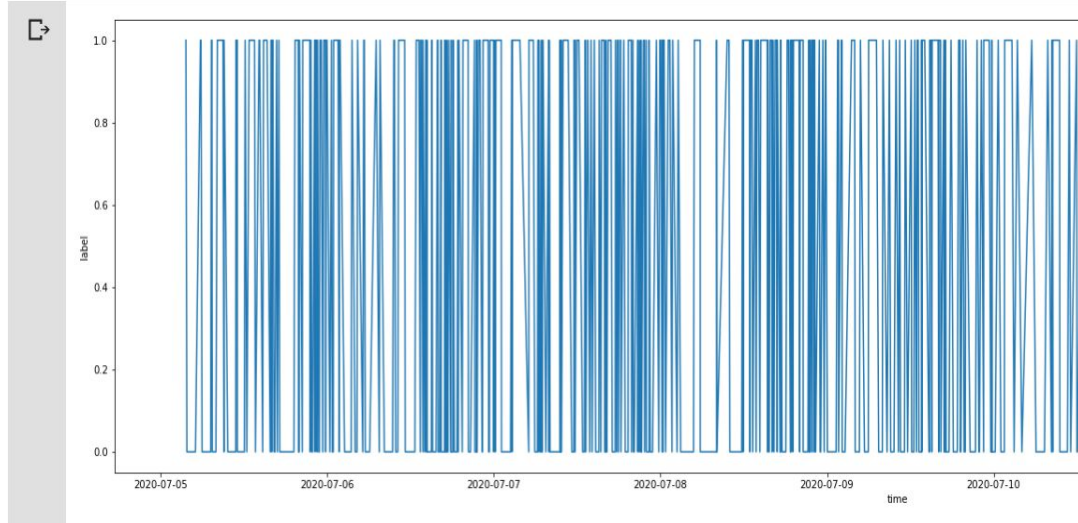

Figure 59: Date-wise fluctuation of positive and negative tweets

Observation:

We can see the rapid and erratic fluctuation from positive to negative with time.

Using pie plots to see the distribution:

```
plt.pie(list(tweet_in.label.value_counts()), labels=['Positive tweets','Negative tweets'],
        labeldistance=0.5,radius= 2.5,startangle=40,colors= ['cyan',"blue"])
plt.show()
```
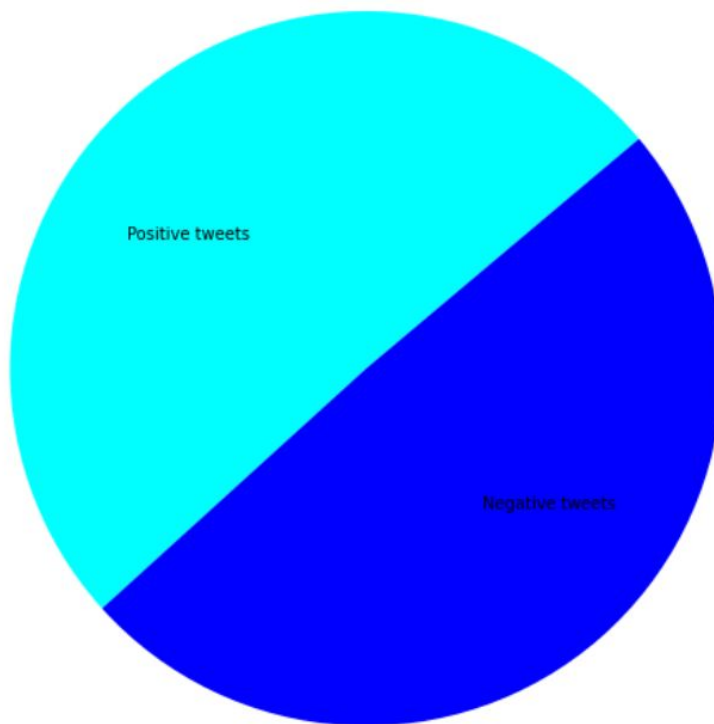

Figure 60: Pie chart for the distribution of tweets

Observations:

The topic "feminist movement" seems to have almost equally divided sentiment over twitter.

CONCLUSION

In conclusion of this case study, a machine learning model using Decision Tree classifier algorithm is built for a client whose purpose is to analyse the public response through sentiment analysis on a particular topic (on twitter platform). Sentiment analysis can be done by brands for market response on certain products, a film or book response, the sentiment around socialist movement etc, can be observed by the client successfully and the generated reports and predictions can be effectively tackled by taking their respective domain related measures accordingly.

Example:

Analysis performed on "feminist movement" in this case study can be used to understand the extent of debatability of the topic and plan news programs and panels accordingly.

REFERENCES

- https://stackoverflow.com/questions/8376691/how-to-remove-hashtag-user-link-of-a-tweet-using-regular-expression
- https://medium.com/@SeoJaeDuk/basic-data-cleaning-engineering-session-twitter-sentiment-data-b9376a91109b
- https://stackoverflow.com/questions/29523254/python-remove-stop-words-from-pandas-dataframe
- https://stackoverflow.com/questions/771918/how-do-i-do-word-stemming-or-lemmatization
- https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html
- https://pypi.org/project/GetOldTweets3/