

MACHINE LEARNING
(MOVIE REVIEWS CLASSIFICATION)

*Summer Internship Report Submitted in partial fulfillment
of the requirement for undergraduate degree of*

Bachelor of Technology
In
COMPUTER SCIENCE AND ENGINEERING

By
KONDA SAI MEGHANA
221710309028

Under the Guidance of



Department Of COMPUTER SCIENCE AND ENGINEERING
GITAM School of Technology

GITAM (Deemed to be University)

Hyderabad-502329

June 2020

DECLARATION

I submit this industrial training work entitled “MOVIE REVIEWS CLASSIFICATION” 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 and Engineering”. I declare that it was carried out independently by me under the guidance of , 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

KONDA SAI MEGHANA

DATE: June2020

221710309028



GITAM (DEEMED TO BE UNIVERSITY)

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled “MOVIE REVIEWS CLASSIFICATION” is being submitted by KONDA SAI MEGHANA (221710309028) 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 2020-21.

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

Assistant Professor
Department of CSE

Professor and HOD
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 completion of this internship.

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

I would like to thank respected Prof. S Phani Kumar, Head of the Department of Computer Science and 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 Mr. 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.

KONDA SAI MEGHANA

221710309028

ABSTRACT

Machine learning algorithms are used to predict the values from the data set by splitting the data set in to train and test and building Machine learning algorithms .The primary task is to build models of higher accuracy .My perception of understanding the given data set has been in the view of undertaking the reviews of various viewers who watch a movie and give their reviews. Based on sentiment analysis under Machine learning, we try to focus our task of sentiment analysis on IMDB movie review databases. We examine the sentiment expression to classify the polarity of the movie review as either positive or negative.A comparative study on different classification algorithms has been performed to determine the most suitable classifier to suit our problem. We conclude that our proposed approach to sentiment classification adds a new feature to the existing features and brings in a progress to future research in this domain. Our approach using classification techniques has the best accuracy of 100% with the logistic regression algorithm and naive bayes algorithm.

Table of Contents:

CHAPTER 1:MACHINE LEARNING.....	10
1.1INTRODUCTION.....	11

1.2IMPORTANCE OF MACHINE LEARNING.....	11
1.3 USES OF MACHINE LEARNING.....	11
1.4 TYPES OF LEARNING ALGORITHMS.....	12
1.4.1 Supervised Learning.....	12
1.4.2 Unsupervised Learning.....	13
1.4.3 Semi Supervised Learning.....	14
1.5 RELATION BETWEEN DATA MINING,MACHINE LEARNING AND DEEP LEARNING.....	14
1.6 DEEP LEARNING.....	15
1.6.1INTRODUCTION.....	15
CHAPTER 2:PYTHON.....	16
2.1 INTRODUCTION TO PYTHON.....	16
2.2 HISTORY OF PYTHON.....	16
2.3 FEATURES OF PYTHON.....	17
2.4 HOW TO SETUP PYTHON.....	17
2.4.1 Installation(using python IDLE).....	17
2.4.2 Installation(using Anaconda).....	18
2.5 PYTHON VARIABLE TYPES.....	22
2.5.1 Python Numbers.....	22
2.5.2 Python Strings.....	23
2.5.3 Python Lists.....	23
2.5.4 Python Tuples.....	23

2.5.5 Python Dictionary.....	24
2.6 PYTHON FUNCTION.....	24
2.6.1 Defining a Function.....	24
2.6.2 Calling a Function.....	25
2.7 PYTHON USING OOPs CONCEPTS.....	25
2.7.1 Class.....	25
2.7.2 __init__ method in class.....	26
CHAPTER 3:CASE STUDY.....	27
3.1 PROBLEM STATEMENT.....	28
3.2 DATA SET.....	28
3.3 OBJECTIVE OF THE CASE STUDY.....	28
CHAPTER 4:MODEL BUILDING.....	29
4.1 PREPROCESSING OF THE DATA.....	29
4.1.1 Getting the Data Set.....	29
4.1.2 Importing the Libraries.....	29
4.1.3 Importing the Data-Set.....	29
4.1.4 Handling the Missing values.....	30
4.1.5 Statistical Analysis.....	31
4.2 BUILD THE MODEL.....	34
4.3 EVALUATING THE CASE STUDY AND TRAINING THE MODEL.....	34

CONCLUSION.....	68
REFERENCES.....	69

LIST OF FIGURES:

Fig1.2 : The Process Flow.....	13
Fig1.4.1 : Unsupervised Learning.....	15
Fig1.4.2 : Semi Supervised Learning.....	16
Fig2.4.1: Python download.....	20
Fig2.4.2 : Anaconda download.....	21
Fig2.4.3 : Jupyter notebook.....	21
Fig2.7.1 : Defining a Class.....	26
Fig4.1.2 : Importing Libraries.....	29
Fig4.1.3 : Reading the Dataset.....	30
Fig4.1.4: Checking null values.....	30
Fig4.1.5.1: Shape of the data.....	30

Fig4.1.5.2:Counts.....	32
Fig4.1.5.3:Countplot.....	32
Fig4.1.5.4:Data description.....	33
Fig4.1.5.5:Output column description.....	33
Fig4.1.5.6:Data information.....	33
Fig4.2.1:Input variable.....	34
Fig4.2.2:Output variables.....	34
Fig4.3.1:Data splitting.....	35
Fig4.3.2:Shape of data after splitting.....	35
Fig4.3.4:Count vectorizer.....	36
Fig4.3.5:Word count.....	36
Fig4.3.6:Tfidf vectorizer.....	37
Fig4.3.7,4.3.8:Tfidf to data.....	37
Fig4.3.9:Obtain the feature names.....	38
Fig4.3.10:Position of the words.....	39
Fig4.3.11:Generate idf of terms.....	39
Figures of NAIVE BAYES ALGORITHM	
Fig4.3.12 - 4.3.23.....	39-50
Figures of LOGISTIC REGRESSION	
Fig4.3.24 - 4.3.34.....	51-55
Figures of RANDOM FOREST CLASSIFICATION	
Fig4.3.35 - 4.3.41.....	56-59

Figures of ROC-AUC:

Fig4.3.42 - 4.3.50.....60-65

CHECKING FOR UNKNOWN DATA.....66

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

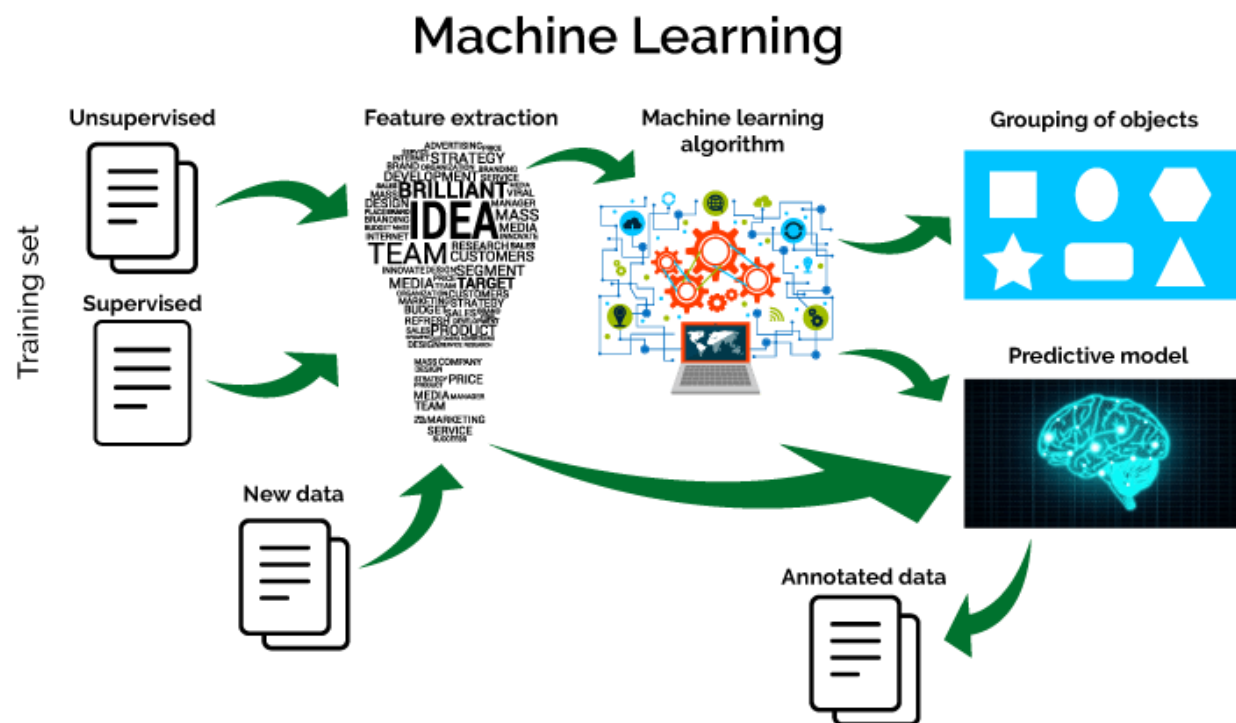


Fig : 1.2-The process Flow

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.

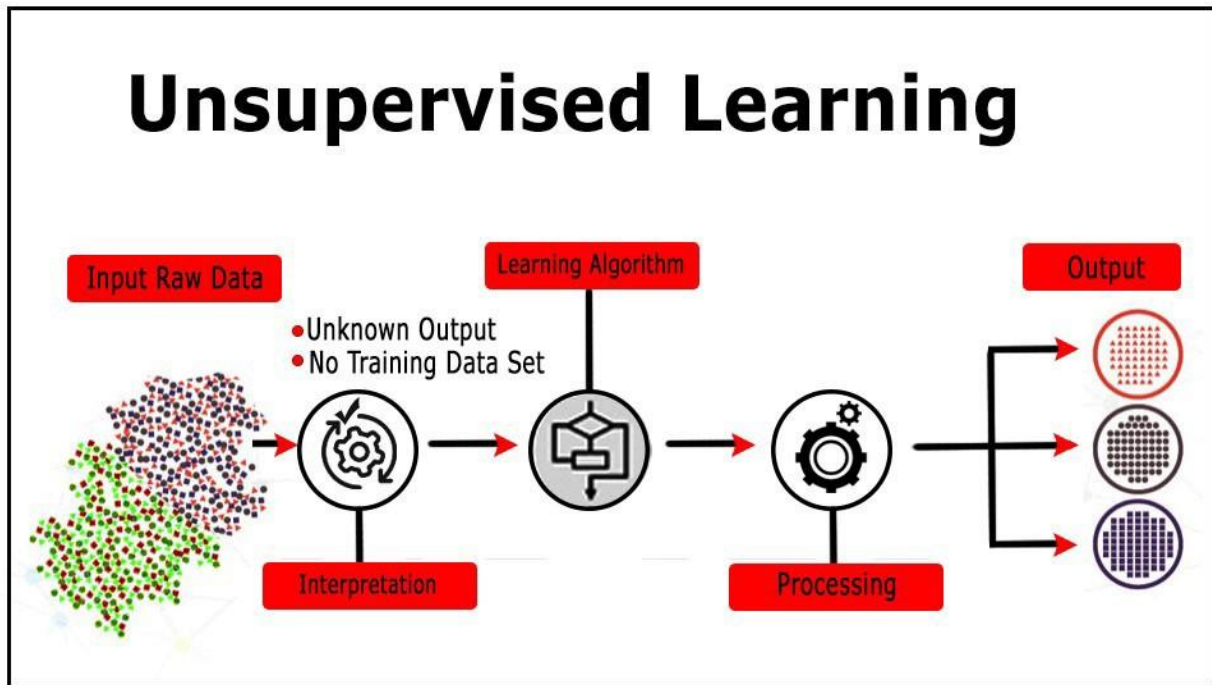


Fig:1.4.1- Unsupervised learning

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.

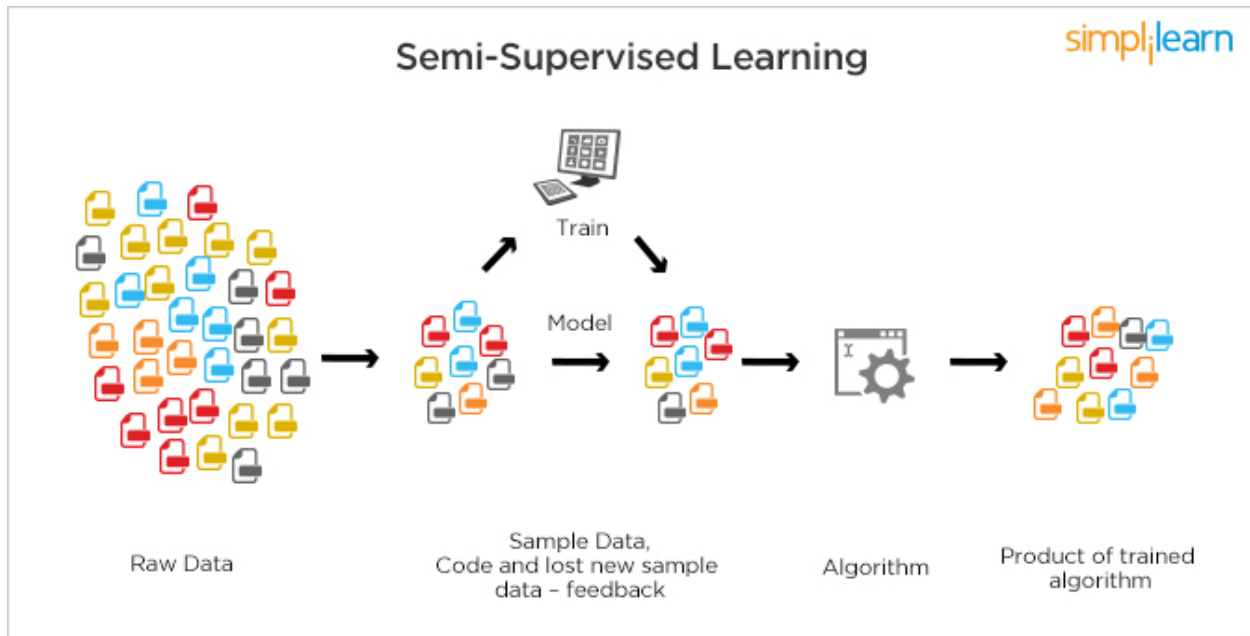


Fig:1.4.2-Semisupervised learning

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.

1.6 DEEP LEARNING :

1.6.1 Introduction

Deep Learning is a sub field of machine learning concerned with algorithms inspired by the structure and function of the brain called artificial neural networks.

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans: learn by example. Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason. It's achieving results that were not possible before.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers.

Deep learning learns from vast amounts of unstructured data that would normally take humans decades to understand and process.

Unstructured data :

The phrase unstructured data usually refers to information that doesn't reside in a traditional row-column database. As you might expect, it's the opposite of structured data — the data stored in fields in a database.

Examples of unstructured data:

Examples of "unstructured data" may include books, journals, documents, metadata, health records, audio, video, analog data, images, files, and unstructured text such as the body of an e-mail message, Web page, or word-processor document.

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.

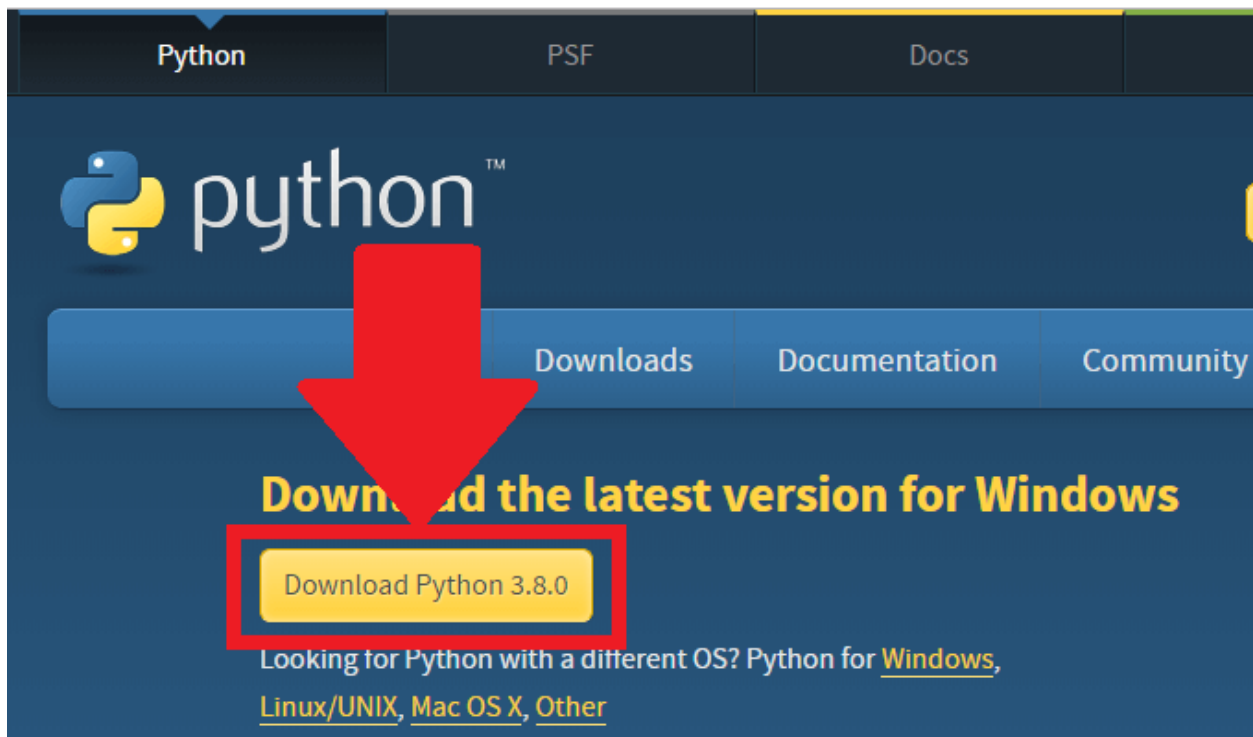


Fig:2.4.1

2.4.2 Installation(using Anaconda):

- Python programs are also executed using Anaconda.EVALUATING THE CASE STUDY
- 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 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)

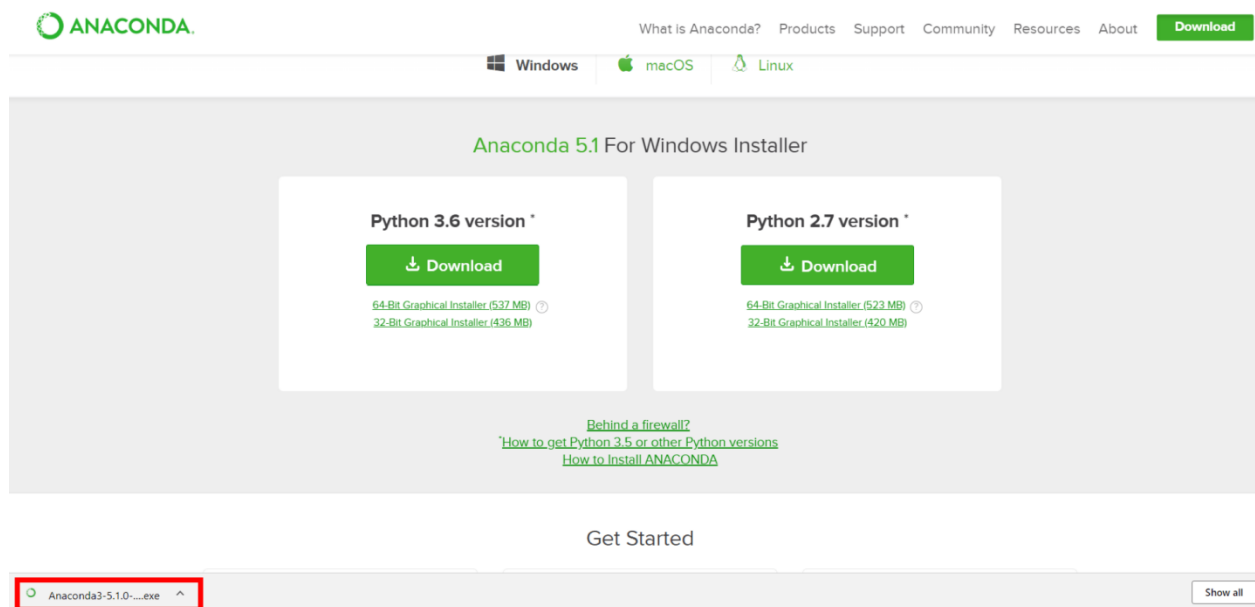


Fig:2.4.2

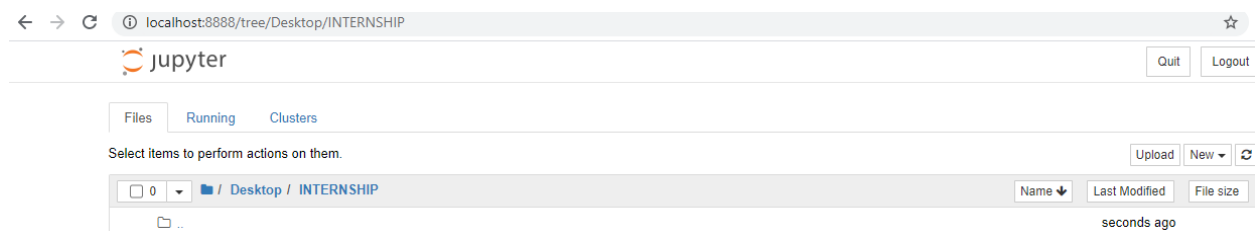


Fig:2.4.3

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 types.
- 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 a 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 `return [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
```

Fig: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

CASE STUDY

PROJECT NAME-MOVIE REVIEWS CLASSIFICATION

Packages used:

- 1.Pandas
- 2.Numpy
3. Matplotlib.pyplot
- 4.Seaborn
- 5.sklearn.model_selection.train_test_split
- 6.sklearn.feature_extraction.text.CountVectorizer

7.sklearn.feature_extraction.text.TfidfVectorizer
7.sklearn.naive_bayes.BernoulliNB
8.sklearn.metrics.classification_report,accuracy_score,confusion_matrix
9.sklearn.linear_model.LogisticRegression
10.sklearn.ensemble.RandomForestClassifier
11.sklearn.metrics.roc_curve,roc_auc_score

Algorithms used:

- 1.NAIVE BAYES ALGORITHM
- 2.LOGISTIC REGRESSION ALGORITHM
- 3.RANDOM FOREST CLASSIFIER ALGORITHM

3.1 PROBLEM STATEMENT:

To classify reviews given by various viewers as a positive review or a negative review after watching a movie using different classification algorithms in Machine Learning and finding the best algorithm.

3.2 DATA SET:

The dataset consists of two columns having column names as REVIEW and SENTIMENT.

The REVIEW column consists of the reviews given by 50000 different viewers after watching a movie.

The SENTIMENT column consists of the sentiments of the viewers classified as either POSITIVE or NEGATIVE.

So, the dataset consists of 50000 rows and 2 columns respectively.

3.3 OBJECTIVE OF THE CASE STUDY:

Movie reviews play an important role in judging whether a movie is good or bad. The reviews predict whether the movie is a success or a failure. A textual movie review tells us the strong and weak points of a movie and deeper insights of a movie review can tell us if the movie in general meets the expectations of the reviewer.

Movie reviews classification using SENTIMENT ANALYSIS aims to extract subjective information from the textual reviews.

CHAPTER 4

MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data involves the following steps:

4.1.1 GETTING THE DATASET:

We can get the data set either from a database or we can get the data from the client.

We can also get the data from Kaggle.

4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirements of the algorithm used.

```
➤ import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Fig:4.1.2.-Importing packages

4.1.3 IMPORTING THE DATA-SET:

Pandas in python provide an interesting method `read_csv()`. The `read_csv` function reads the entire dataset from a comma separated values (CSV) file and we can assign it to a DataFrame to which all the operations can be performed. It helps us to access each and every row as well as columns and each and every value can be accessed using the dataframe. Any missing value or NaN value has to be cleaned.

The dataset has to be downloaded into the system and save it so as to read it.

Importing and reading the dataset :

```
data = pd.read_csv("imdb_dataset.csv")
data.head()
```

	review	sentiment
0	One of the other reviewers has mentioned that ...	positive
1	A wonderful little production. The...	positive
2	I thought this was a wonderful way to spend ti...	positive
3	Basically there's a family where a little boy ...	negative
4	Petter Mattei's "Love in the Time of Money" is...	positive

Fig: 4.1.3-Reading the data

4.1.4 HANDLING MISSING VALUES:

We should always check if our dataset has any missing values before building the model.

If we find any missing values , we need to impute them using different techniques such as:

`dropna()`,`fillna()`,`interpolate()` or either using mean,median imputations.

To check for any missing values :

```
data.isnull().sum()
```

```
review      0  
sentiment   0  
dtype: int64
```

Fig:4.1.4-Checking for null values

Since there were no missing values, we can proceed with the further steps.

To visualize the missing values, we can use a heatmap from the SEABORN library.

```
sns.heatmap(data.isnull())
```

```
]: <matplotlib.axes._subplots.AxesSubplot at 0x18be0da7cc0>
```

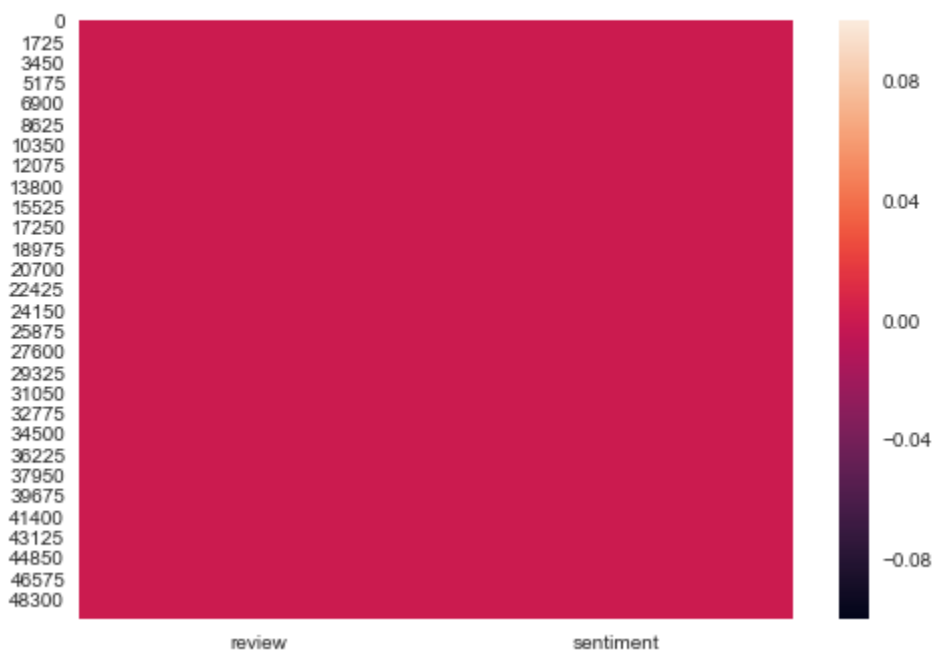


Fig:4.1.4(a)- Heatmap for null values

4.1.5 STATISTICAL ANALYSIS :

To check the number of rows and columns in the dataset:

```
data.shape  
:  
(50000, 2)
```

Fig: 4.1.5.1-for shape of the data

We need to check the features of the SENTIMENT column :

```
data.sentiment.value_counts()  
:  
negative    25000  
positive    25000  
Name: sentiment, dtype: int64
```

Fig:4.1.5.2-counts

Visualization of the target column i.e. the SENTIMENT column using the COUNTPLOT in the ‘matplotlib’ library.

```
plt.title("MOVIE REVIEWS")
sns.set(style="darkgrid")
sns.countplot(data.sentiment,palette=["blue","red"])
plt.legend

]: <function matplotlib.pyplot.legend(*args, **kwargs)>
```

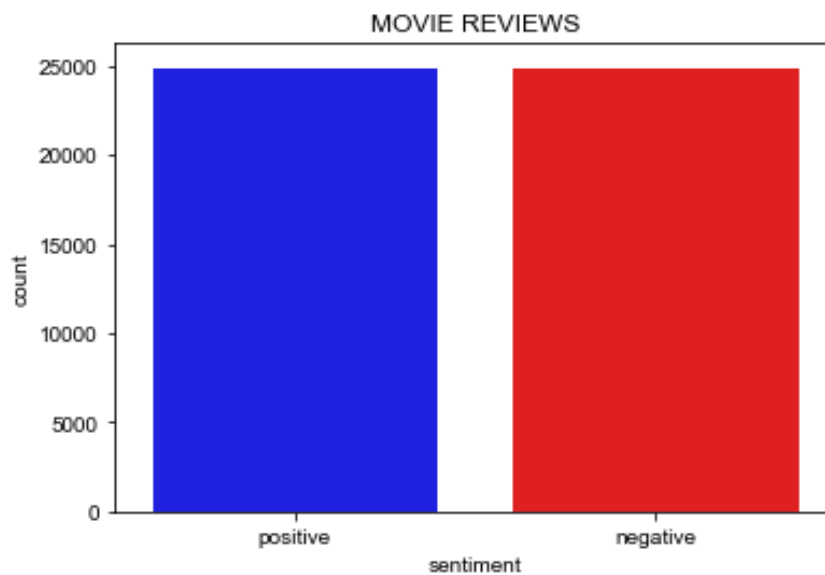


Fig:4.1.5.3-countplot

To check the description of the data:

```
data.describe()
```

```
]:
```

	review	sentiment
count	50000	50000
unique	49582	2
top	Loved today's show!!! It was a variety and not...	negative
freq	5	25000

Fig:4.1.5.4-data description

To check the description of the target column:

```
data['sentiment'].describe()
: count      50000
  unique        2
  top      negative
  freq      25000
  Name: sentiment, dtype: object
```

Fig:4.1.5.5- output description

To get the information about the data:

```
data.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
review      50000 non-null object
sentiment   50000 non-null object
dtypes: object(2)
memory usage: 781.3+ KB
```

Fig:4.1.5.6-data information

4.2 BUILD THE MODEL:

Now we separate the input column-REVIEW and output column(target column)-SENTIMENT.

We take a variable 'X' and store the input data into it.

```

X = data.review
X.head()

]: 0    One of the other reviewers has mentioned that ...
   1    A wonderful little production. <br /><br />The...
   2    I thought this was a wonderful way to spend ti...
   3    Basically there's a family where a little boy ...
   4    Petter Mattei's "Love in the Time of Money" is...
   Name: review, dtype: object

```

Fig:4.2.1-input variable

We then take another variable 'y' and store the output data into it:

```

y= data.sentiment
y.head()

0    positive
1    positive
2    positive
3    negative
4    positive
   Name: sentiment, dtype: object

```

Fig:4.2.2-outpunt variable

4.3 EVALUATING THE CASE STUDY,TRAINING THE MODEL:

After preprocessing the data, we then need to train the data . For that purpose, first we need to split the data using the 'train_test_split ' , by importing it from the scikit learn library we have the 'model_selection' package that consists of the 'train_test_split ' method.

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'.

- training set - a subset to train a model.(Model learns patterns between Input and Output)
- test set - a subset to test the trained model.(To test whether the model has correctly learnt)

```
# split the data
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.2,random_state=5) #(input,output,testsize)
```

Fig: 4.3.1- data splitting

We need to specify the test size along with passing the input and output.

The amount or percentage of Splitting can be taken as specified (i.e. train data = 75% , test data =25% or train data = 80% , test data= 20%)

Here we have taken the test size as 0.2 indicating train data =80% and test data=20%

Next we check the shape of the training input ,training output, testing input,testing output.

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

```
(40000,)
(40000,)
(10000,)
(10000,)
```

Fig:4.3.2-shape of splitted data

In order to use textual data for predictive modeling, the text must be parsed to remove certain words – this process is called TOKENIZATION. It also enables the pre-processing of text data prior to generating the vector representation.

COUNT VECTORIZATION :

The CountVectorizer provides a simple way to both tokenize a collection of text documents and build a vocabulary of known words, but also to encode new documents using that vocabulary.

You can use it as follows:

1. Create an instance of the CountVectorizer class.
2. Call the fit() function in order to learn a vocabulary from one or more documents.
3. Call the transform() function on one or more documents as needed to encode each as a vector.

Scikit-learn's **CountVectorizer** is used to convert a collection of text documents to a vector of term/token counts. It also enables the pre-processing of text data prior to generating the vector representation. This functionality makes it a highly flexible feature representation module for text.

```
# COUNT VECTORIZER
from sklearn.feature_extraction.text import CountVectorizer
# creating an object
count_vect = CountVectorizer()
```

Fig:4.3.4-count vectorizer

Here we created an object CountVectorizer() and stored it in a variable called count_vect.

```
In [15]: # Generate the word counts for the words in the documents
word_count_vector = count_vect.fit(X_train)

# to get the feature names
word_count_vector.get_feature_names()

Out[15]: ['00',
'000',
'0000000000001',
'00000001',
'00001',
'00015',
'000dm',
'000s',
'001',
'003830',
'006',
'0069',
'007',
'0079',
'007s',
'0080',
'0083',
'0093638',
'00am',
'001']
```

Fig:4.3.5-word count

Here we have fitted the input data inside the count_vect variable and stored it in another variable word_count_vect.

Then we have displayed the different feature names from our data.

TFIDF VECTORIZER:

An alternative is to calculate word frequencies, and by far the most popular method is called TF-IDF. This is an acronym that stands for “Term Frequency – Inverse Document” Frequency which are the components of the resulting scores assigned to each word.

- **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. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

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.

The **term frequency**, the number of times a term occurs in a given document, is multiplied with **idf** component, which is computed as **idf** is the inverse document frequency, so it's the ratio of the number of documents (all documents vs documents that contain the term at least once).

TF-IDF stands for “Term Frequency — Inverse Document Frequency”.

This is a technique to quantify a word in documents, we generally compute a weight to each word which signifies the importance of the word in the document and corpus.

```
#TFIDF VECTORIZER
from sklearn.feature_extraction.text import TfidfVectorizer
#initialize an object for the tfidf vectorizer
tfidf = TfidfVectorizer()
```

Fig:4.3.6-tfidf vectorizer

We have created an object TfidfVectorizer() and stored it in the 'tfidf' variable.

```
#apply the tfidf to the data(X_train)

X_train_transformed = tfidf.fit_transform(X_train)
X_train_transformed

<40000x92916 sparse matrix of type '<class 'numpy.float64'>'
  with 5463017 stored elements in Compressed Sparse Row format>
```

Fig:4.3.7-tfidf to data

Here first we apply the tfidf to the training input data(X_train) and store it in X_train_transformed.

```
#apply the tfidf to the data(X_test)
X_test_transformed = tfidf.transform(X_test)
X_test_transformed

<10000x92916 sparse matrix of type '<class 'numpy.float64'>'
  with 1353926 stored elements in Compressed Sparse Row format>
```

Fig:4.3.8-tfidf to data

Here we applied the tfidf to the testing input data(X_test) and stored it in X_test_transformed.

Therefore, now our input for training data is 'X_train_transformed' and its corresponding output is 'y_train'.

Similarly, 'X_test_transformed' is the input for testing data and its respective output will be 'y_test'.

```
In [19]: #Feature names  
         tfidf.get_feature_names()  
  
Out[19]: ['00',  
          '000',  
          '0000000000001',  
          '00000001',  
          '00001',  
          '00015',  
          '000dm',  
          '000s',  
          '001',  
          '003830',  
          '006',  
          '0069',  
          '007',  
          '0079',  
          '007s',  
          '0080',  
          '0083',  
          '0093638',  
          '00am',  
          '001']
```

Fig:4.3.9 - To obtain the feature names

```
In [20]: # position of the words
tfidf.vocabulary_

Out[20]: {'there': 82411,
          'is': 42612,
          'an': 3971,
          'excellent': 28243,
          'reason': 66926,
          'edison': 25890,
          'went': 90027,
          'straight': 78734,
          'to': 83270,
          'video': 88387,
          'it': 42740,
          'would': 91448,
          'have': 37220,
          'landed': 46662,
          'in': 40759,
          'theaters': 82306,
          'with': 90990,
          'crumbling': 19627,
          'thud': 82776,
          '...': 82305}
```

Fig:4.3.10 - to obtain the position of the words

```
# Idf of the terms
tfidf.idf_

array([ 6.67667881,  5.72736282, 10.90351255, ..., 10.90351255,
        10.90351255, 10.90351255])
```

Fig: 4.3.11 - to generate the idf of the terms

METRICS USED IN THE ALGORITHMS:

To check the best model, we need to find out the different metrics on which the model will be tested. They are:

CONFUSION MATRIX :

It is a performance measurement for machine learning classification problems where output can be two or more classes. It is a table with 4 different combinations of predicted and actual values.

It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC Curve.

True Positive (TP): It refers to the number of predictions where the classifier correctly predicts the positive class as positive.

True Negative (TN): It refers to the number of predictions where the classifier correctly predicts the negative class as negative.

False Positive (FP): It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive.

When a negative sample is falsely classified as a Positive, it is called a False Positive(FP)

False Negative (FN): It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.

When a Positive sample is falsely classified as Negative, we call this a False Negative (FN)

Precision(Positive Predicted Value):

We need to look at the total number of predicted Positives (the True Positives plus the False Positives, $TP+FP$), and see how many of them are True Positive (TP).

Recall:

Recall means what proportion of actual Positives is correctly classified.

Accuracy:

Accuracy means what proportion of both Positive and Negative were correctly classified

The general formula for accuracy is $(TP+TN)/(TP+TN+FP+FN)$.

F1-Score:

We would like to summarize the models' performance into a single metric. That's where F1-score is used. It's a way to combine precision and recall into a single number. F1-score is computed using a mean ("average"), but not the usual arithmetic mean.



$$Accuracy = \frac{T_p + T_n}{T_p + T_n + F_p + F_n}$$
$$Precision = \frac{T_p}{T_p + F_p}$$
$$Recall = \frac{T_p}{T_p + T_n}$$
$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Fig:4.4

Rate is a measure factor in a confusion matrix. It has also 4 type TPR, FPR, TNR, FNR

For better performance, **TPR**, **TNR** should be high and **FNR**, **FPR** should be low.

True Positive Rate :

$$\text{True Positive Rate} = \text{Sensitivity} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Sensitivity tells us what proportion of the positive class got correctly classified.

False Negative Rate :

$$FNR = \frac{FN}{TP + FN}$$

False Negative Rate (FNR) tells us what proportion of the positive class got incorrectly classified by the classifier.

A higher TPR and a lower FNR is desirable since we want to correctly classify the positive class.

True Negative Rate(Specificity):

$$Specificity = \frac{TN}{TN + FP}$$

Specificity/TNR tells us what proportion of the negative class got correctly classified.

False Positive Rate:

$$\text{False Positive Rate} = (1 - \text{Specificity}) = \frac{\text{False Positives}}{\text{False Positives} + \text{True Negatives}}$$

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

A higher TNR and a lower FPR is desirable since we want to correctly classify the negative class.

Out of these metrics, Sensitivity and Specificity are perhaps the most important

CLASSIFICATION REPORT:

A Classification report is used to measure the quality of predictions from a classification algorithm. The report shows the main classification metrics precision, recall and f1-score, support on a per-class basis. The metrics are calculated by using true and false positives, true and false negatives.

- The recall means "how many of this class you find over the whole number of element of this class"
- The precision will be "how many are correctly classified among that class"

- The f1-score is the harmonic mean between precision & recall
- The support is the number of occurrence of the given class in your dataset

ROC CURVE:

An ROC curve (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.

AUC :

Area under the ROC Curve :

AUC stands for "Area under the ROC Curve." That is, **AUC** measures the entire two-dimensional area underneath the entire ROC curve.

Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test.

Any classifier can be used to create an ROC curve, but it must be able to fit and predict_proba.

We now use different algorithms to train and predict the data and find out the optimum algorithm that best fits our data and gives maximum accuracy.

We use :

1. NAIVE BAYES ALGORITHM
2. LOGISTIC REGRESSION ALGORITHM
3. RANDOM FOREST CLASSIFICATION ALGORITHM

1.NAIVE BAYES ALGORITHM:

Naive Bayes Algorithm comes under Supervised Learning. It is a classification algorithm, which performs well on numerical and the text data. It is not a single algorithm but a family of algorithms where all of them share a common principle, i.e. every pair of features being classified is independent of each other.

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. It is one of the simplest supervised learning algorithms. Naive Bayes classifier is the fast, accurate and reliable algorithm. Naive Bayes classifiers have high accuracy and speed on large datasets.

Naive Bayes Classifier assumes that the effect of a particular feature in a class is independent of other features. For example, a loan applicant is desirable or not depending on his/her income, previous loan and transaction history, age, and location.

Even if these features are interdependent, these features are still considered independently. This assumption simplifies computation, and that's why it is considered as 'Naive'. This assumption is called class conditional independence.

First we import the 'BernoulliNB' method from 'sklearn.naive_bayes' package .

Then we create an object called BernoulliNB() and store it in a variable called 'model_BernNB'

```
# Apply the naive bayes algorithm
from sklearn.naive_bayes import BernoulliNB
#creating an object
model_BernNB = BernoulliNB()
```

Fig:4.3.12

Next we apply the algorithm to the data and fit the data using the syntax:

objectname.fit(input,output)

```
▶ # applying the algorithm to the data
  # objectname.fit(input,output)

model_BernNB.fit(X_train_transformed,y_train)

: BernoulliNB()
```

Fig:4.3.13

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

For training data :

Here we have given the input data for prediction and stored it in a variable called 'y_train_pred1'

```
# prediction on train data
# syntax: objectname.predict(input)
y_train_pred1 = model_BernNB.predict(X_train_transformed)
```

Fig:4.3.14

For comparing the actual values with model-predicted values,we generate a report called 'classification_report', 'accuracy_score','confusion_matrix'.

We need to import them from the 'sklearn.metrics' package.

```
# compare the actual values(y_train) with predicted values(y_train_pred)
from sklearn.metrics import confusion_matrix , classification_report,accuracy_score
confusion_matrix(y_train,y_train_pred1)
```

Fig: 4.3.15

```
print(classification_report(y_train,y_train_pred1))
```

	precision	recall	f1-score	support
negative	0.88	0.92	0.90	20100
positive	0.92	0.87	0.90	19900
accuracy			0.90	40000
macro avg	0.90	0.90	0.90	40000
weighted avg	0.90	0.90	0.90	40000

Fig: 4.3.16

```
accuracy_score(y_train,y_train_pred1)
```

0.8984

Fig:4.3.17

To visualize the confusion matrix , we use the heatmap to get a clear and a better understanding:

```
sns.heatmap(confusion_matrix(y_train,y_train_pred1),annot=True,fmt='3.0f',annot_kws={'size':'20'})
<matplotlib.axes._subplots.AxesSubplot at 0x18bc050f5c0>
```

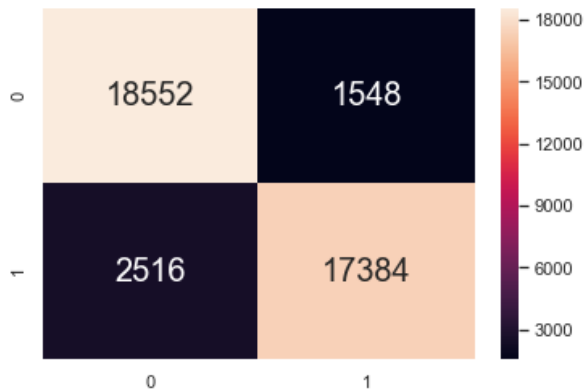


Fig: 4.3.18

For testing data :

Here we have given the input data for prediction and stored it in a variable called 'y_test_pred1'

```
# prediction on test data
# syntax: objectname.predict(input)
y_test_pred1 = model_BernNB.predict(X_test_transformed)
```

Fig:4.3.19

```
# compare the actual values(y_test) with predicted values(y_test_pred)
confusion_matrix(y_test,y_test_pred1)

: array([[4317,  583],
        [ 841, 4259]], dtype=int64)
```

Fig:4.3.20


```
print(classification_report(y_test,y_test_pred1))
```

	precision	recall	f1-score	support
negative	0.84	0.88	0.86	4900
positive	0.88	0.84	0.86	5100
accuracy			0.86	10000
macro avg	0.86	0.86	0.86	10000
weighted avg	0.86	0.86	0.86	10000

Fig:4.3.21

```
accuracy_score(y_test,y_test_pred1)
```

0.8576

Fig:4.3.22

```
sns.heatmap(confusion_matrix(y_test,y_test_pred1),annot=True,fmt='3.0f',annot_kws={'size':'20'})
```

<matplotlib.axes._subplots.AxesSubplot at 0x18bd3506438>

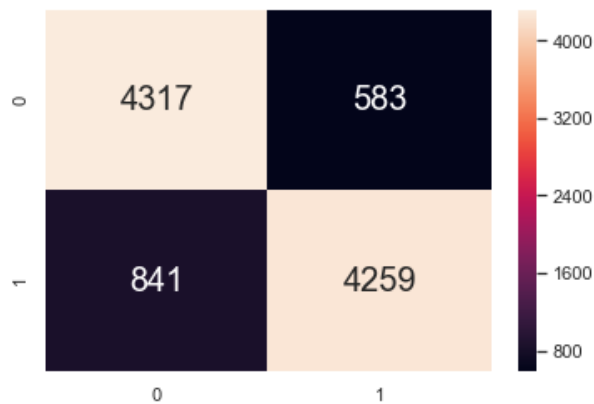


Fig:4.3.23

After predicting on both training and testing data, we have obtained an accuracy of 0.89 for training data and an accuracy of 0.85 for testing data.

We now check the accuracies of other algorithms and get the best one.

2.LOGISTIC REGRESSION 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 dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis and it predicts the probability.

Example: Positive or Negative ,Yes or Not, pass or fail, defective or non-defective, etc.

Also called a classification algorithm, because we are classifying the data. It predicts the probability associated with each dependent variable category.

Logistic regression can be binomial, ordinal or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types, "positive" and "negative" (which may represent, for example, "dead" vs. "alive" or "win" vs. "loss").

Multinomial Logistic Regression deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C") that are not ordered.

Ordinal Logistic Regression deals with dependent variables that are ordered.

First we import the 'LogisticRegression' method from the 'sklearn.linear_model' package .

Then we create an object called LogisticRegression() and store it in a variable called 'reg'

Next we apply the algorithm to the data and fit the data using the syntax:

“objectname.fit(input,output)”

```
# build the classifier on training data
#sklearn library: import,instantiate,fit
from sklearn.linear_model import LogisticRegression
reg = LogisticRegression()
reg.fit(X_train_transformed,y_train) #input and output will be passed to fit method
LogisticRegression()
```

Fig:4.3.24

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

For training data :

Here we have given the input data for prediction and stored it in a variable called 'y_train_pred2'

```
#predicting on train data
# syntax : objectname.predict(Input)
y_train_pred2 =reg.predict(X_train_transformed)
y_train_pred2

array(['negative', 'positive', 'negative', ..., 'positive', 'negative',
       'positive'], dtype=object)
```

Fig:4.3.25

For comparing the actual values with model-predicted values, we generate a report called 'classification_report', 'accuracy_score', 'confusion_matrix'.

We need to import them from the 'sklearn.metrics' package.

```
# confusion matrix for the training data
# confusion matrix(actual values, predicted values)
from sklearn.metrics import confusion_matrix, accuracy_score
conf = confusion_matrix(y_train, y_train_pred2)
conf

array([[18630, 1470],
       [ 1248, 18652]], dtype=int64)
```

Fig: 4.3.26

```
from sklearn.metrics import classification_report
print(classification_report(y_train, y_train_pred2))
```

	precision	recall	f1-score	support
negative	0.94	0.93	0.93	20100
positive	0.93	0.94	0.93	19900
accuracy			0.93	40000
macro avg	0.93	0.93	0.93	40000
weighted avg	0.93	0.93	0.93	40000

Fig:4.3.27

```
from sklearn.metrics import accuracy_score
accuracy_score(y_train, y_train_pred2)

0.93205
```

Fig:4.3.28

To visualize the confusion matrix using the heatmap.

```
sns.heatmap(confusion_matrix(y_train,y_train_pred2),annot=True,fmt='3.0f',annot_kws={'size':'20'})  
<matplotlib.axes._subplots.AxesSubplot at 0x18bcea0e438>
```

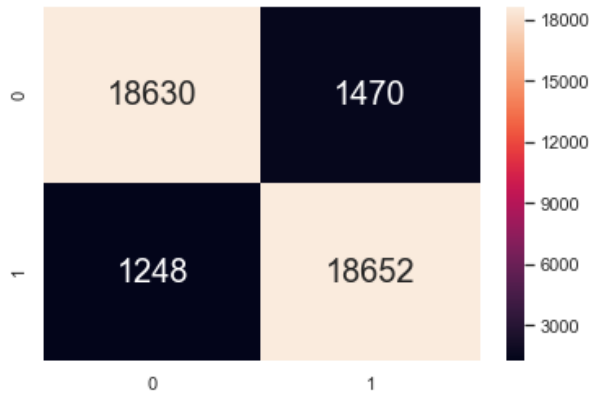


Fig:4.3.29

For testing data:

Here we have given the input data for prediction and stored it in a variable called 'y_test_pred2'

```
#predicting on testing data  
y_test_pred2 =reg.predict(X_test_transformed)  
y_test_pred2  
: array(['positive', 'positive', 'positive', ..., 'negative', 'positive',  
       'positive'], dtype=object)
```

Fig:4.3.30

For comparing the actual values with model-predicted values,we generate a report called 'classification_report', 'accuracy_score', 'confusion_matrix'.

We need to import them from the 'sklearn.metrics' package.

```

conf = confusion_matrix(y_test,y_test_pred2)
conf

array([[4341,  559],
       [ 468, 4632]], dtype=int64)

```

Fig:4.3.31

```

print(classification_report(y_test,y_test_pred2))

```

	precision	recall	f1-score	support
negative	0.90	0.89	0.89	4900
positive	0.89	0.91	0.90	5100
accuracy			0.90	10000
macro avg	0.90	0.90	0.90	10000
weighted avg	0.90	0.90	0.90	10000

Fig:4.3.32

```

accuracy_score(y_test,y_test_pred2)

0.8973

```

Fig:4.3.33

To visualize the confusion matrix, we use the heatmap.

```
sns.heatmap(confusion_matrix(y_test,y_test_pred2),annot=True,fmt='3.0f',annot_kws={'size':'20'})  
<matplotlib.axes._subplots.AxesSubplot at 0x18bd323c6a0>
```



Fig:4.3.34

After building and predicting the model on both training data and testing data, we got an accuracy of 0.93 and 0.89 respectively.

3.RANDOM FOREST CLASSIFICATION ALGORITHM:

Random forest is a supervised learning algorithm. The "forest" it builds is an ensemble of decision trees, usually trained with the “bagging” method. The general idea of the bagging method is that a combination of learning models increases the overall result.

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks).

First we import the ‘RandomForestClassifier’ method from the ‘sklearn.ensemble’ package .

Then we create an object called RandomForestClassifier() and store it in a variable called 'rfc'

Next we apply the algorithm to the data and fit the data using the syntax:

objectname.fit(input,output).

```
#Import, intialize and fit  
#Import the RfC from sklearn  
from sklearn.ensemble import RandomForestClassifier  
  
#Initilaize the object for RFC  
rfc = RandomForestClassifier(n_estimators = 40)  
  
#fit the RFC to the dataset  
rfc.fit(X_train_transformed,y_train)  
  
RandomForestClassifier(n_estimators=40)
```

Fig.4.3.35

Next , we need to predict the data for both training data and testing data and compare the actual value(y_train and y_test) with the model predicted values.

For training data :

Here we have given the input data for prediction and stored it in a variable called 'y_train_pred3'.

Then we obtained a classification report and a confusion matrix to get the optimum values.


```
#Prediction on training data
#Syntax: ojectname.predict(InputValues)
y_train_pred3 = rfc.predict(X_train_transformed)

from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_train, y_train_pred3))
```

	precision	recall	f1-score	support
negative	1.00	1.00	1.00	20100
positive	1.00	1.00	1.00	19900
accuracy			1.00	40000
macro avg	1.00	1.00	1.00	40000
weighted avg	1.00	1.00	1.00	40000

Fig.4.3.36

```
accuracy_score(y_train,y_train_pred3)
```

1.0

Fig.4.3.37

To visualize the confusion matrix, we use the heatmap.

```
sns.heatmap(confusion_matrix(y_train,y_train_pred3),annot=True,fmt='3.0f',annot_kws={'size':'20'})
```

<matplotlib.axes._subplots.AxesSubplot at 0x18bd52a93c8>



Fig:4.3.38

For testing data:

Here we have given the input data for prediction and stored it in a variable called 'y_test_pred3'

For comparing the actual values with model-predicted values, we generate a report called 'classification_report', 'accuracy_score', 'confusion_matrix'.

We need to import them from the 'sklearn.metrics' package.

```
#Prediction on testing data
y_test_pred3 = rfc.predict(X_test_transformed)
print(classification_report(y_test, y_test_pred3))
```

	precision	recall	f1-score	support
negative	0.81	0.84	0.83	4900
positive	0.84	0.81	0.82	5100
accuracy			0.82	10000
macro avg	0.83	0.83	0.82	10000
weighted avg	0.83	0.82	0.82	10000

Fig:4.3.39

```
accuracy_score(y_test, y_test_pred3)

0.8248
```

Fig:4.3.40

```
sns.heatmap(confusion_matrix(y_test,y_test_pred3),annot=True,fmt='3.0f',annot_kws={'size':'20'})  
<matplotlib.axes._subplots.AxesSubplot at 0x272d25ea860>
```

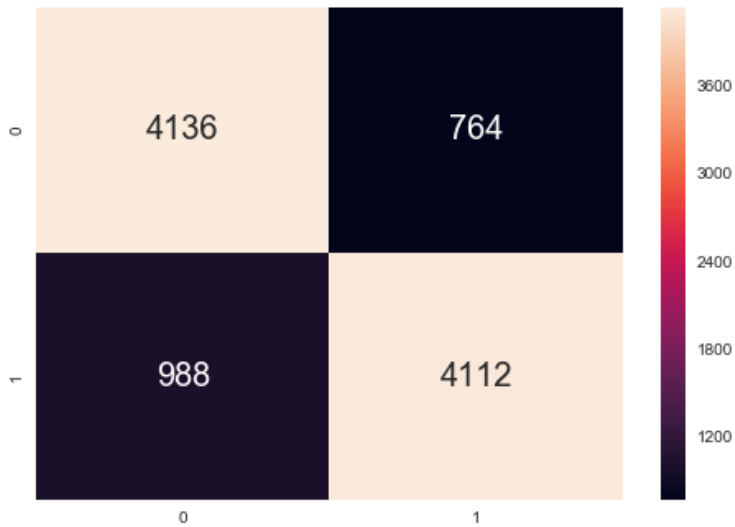


Fig.4.3.41

After building and predicting the model on both training data and testing data, we got an accuracy of 1.0 and 0.82 respectively.

After building the model using the above three algorithms, to find the best fit among the three, we visualize the data using the roc-curve and auc_score.

ROC curve:

An ROC curve (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.

AUC :

Area under the ROC Curve :

AUC stands for "Area under the ROC Curve." That is, **AUC** measures the entire two-dimensional area underneath the entire ROC curve.

Accuracy is measured by the area under the ROC curve. An area of 1 represents a perfect test; an area of 0.5 represents a worthless test.

Any classifier can be used to create an ROC curve, but it must be able to fit and predict_proba.

Here , we use:

sklearn.naive_bayes.BernoulliNB,

sklearn.linear_model.LogisticRegression,

sklearn.ensemble.RandomForestClassifier

```
#training the models
from sklearn.naive_bayes import BernoulliNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier

#naive bayes, logistic regression, random forest
model_BernNB = BernoulliNB()
reg = LogisticRegression()
rfc = RandomForestClassifier(n_estimators = 40)

#fitting the models
model_BernNB.fit(X_train_transformed,y_train)
reg.fit(X_train_transformed,y_train)
rfc.fit(X_train_transformed,y_train)

#predicting the probabilities
pred_prob1=model_BernNB.predict_proba(X_test_transformed)
pred_prob2=reg.predict_proba(X_test_transformed)
pred_prob3=rfc.predict_proba(X_test_transformed)
```

Fig:4.3.42

Here, we find out the predicted probabilities using 'predict_proba' and store them in 'pred_prob1', 'pred_prob2', 'pred_prob3' for the respective three models.

Next, we import the 'roc_curve' from the 'sklearn.metrics' package.

And pass the fpr, tpr, threshold values into them.

```
from sklearn.metrics import roc_curve

# roc curve for models
fpr1, tpr1, thresh1 = roc_curve(y_test_pred1, pred_prob1[:,1], pos_label='positive')
fpr2, tpr2, thresh2 = roc_curve(y_test_pred2, pred_prob2[:,1], pos_label='positive')
fpr3, tpr3, thresh3 = roc_curve(y_test_pred3, pred_prob3[:,1], pos_label='positive')
```

Fig:4.3.43

Here we have taken the pos_label value as "positive" to obtain the predictions on the 'positive' attribute in the output column.

```
from sklearn.metrics import roc_auc_score

# auc scores
auc_score1 = roc_auc_score(y_test_pred1, pred_prob1[:,1])
auc_score2 = roc_auc_score(y_test_pred2, pred_prob2[:,1])
auc_score3 = roc_auc_score(y_test_pred3, pred_prob3[:,1])

print(auc_score1, auc_score2, auc_score3)

1.0 1.0 0.9205980969104504
```

Fig:4.3.44

This returns the False Positive rate fpr, the True Positive rate tpr, and thresholds, the thresholds used to decide tps. To find the AUC of the ROC curve, use sklearn.metrics.roc_auc_score(y_true, y_score), returning a float between 0 and 1. This

shows a "perfect" ROC curve, where the AUC is 1. This means that the classifier correctly predicts each test sample.

```
import matplotlib.pyplot as plt
plt.style.use('seaborn')

#plot roc curves
plt.plot(fpr1, tpr1, linestyle='--', color='orange', label='Naive Bayes')
plt.plot(fpr2, tpr2, linestyle='--', color='green', label='Logistic Regression')
plt.plot(fpr3, tpr3, linestyle='--', color='blue', label='Random Forest Classification')

#title
plt.title('ROC CURVE')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc='best')
plt.savefig('ROC', dpi=300)
plt.show()
```

Fig:4.3.45

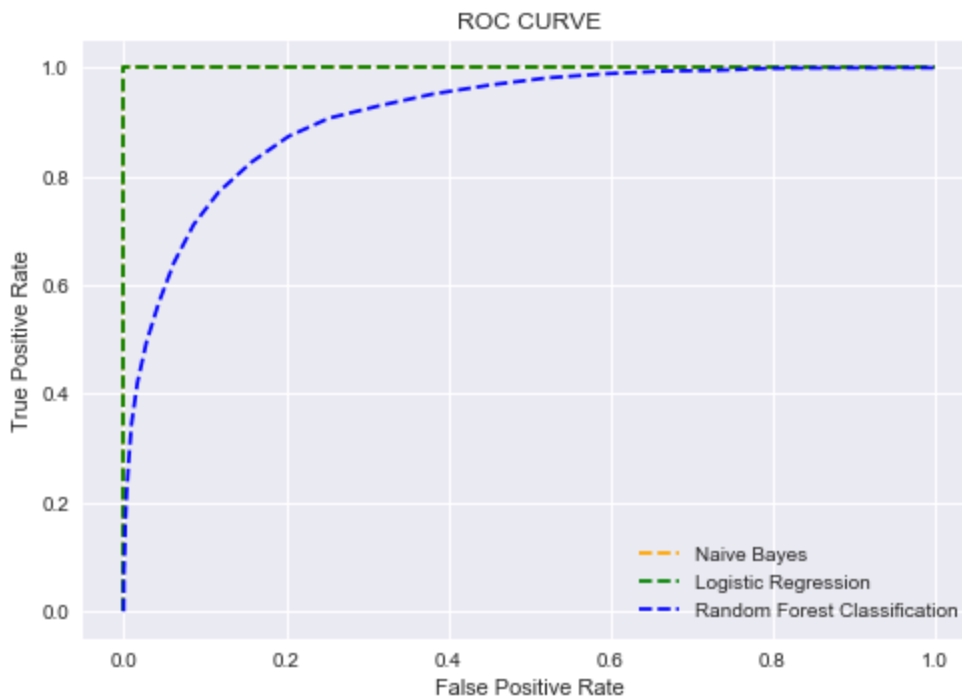


Fig:4.3.46

After plotting the ROC-AUC curve, it is observed that we got 1.0 as accuracy at **TRUE POSITIVE RATE(TPR)** for both naive bayes and logistic regression. Therefore, the graphs for naive bayes and logistic regression overlap at 1.0.

So, the best models are naive bayes and logistic regression.

Visualizations of fpr and tpr of the three models:

For NAIVE BAYES:

```
#visualization for naive bayes(accuracy)  
plt.plot(fpr1,tpr1)
```

```
[<matplotlib.lines.Line2D at 0x1a9eb8f2d30>]
```

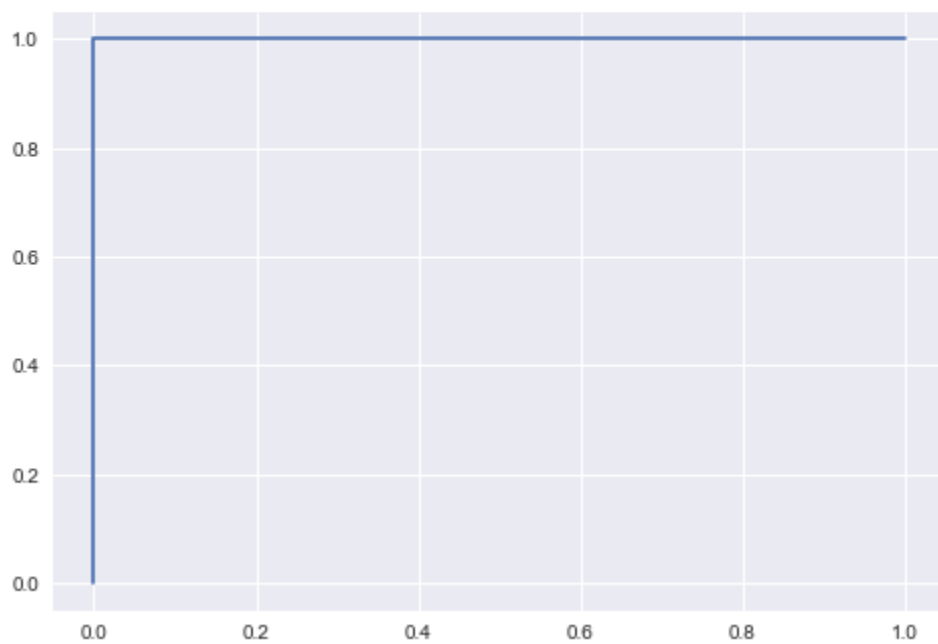


Fig:4.3.47

For LOGISTIC REGRESSION:

```
#visualization for logistic regression(accuracy)  
plt.plot(fpr2,tpr2)
```

```
[<matplotlib.lines.Line2D at 0x1a9eb946e48>]
```

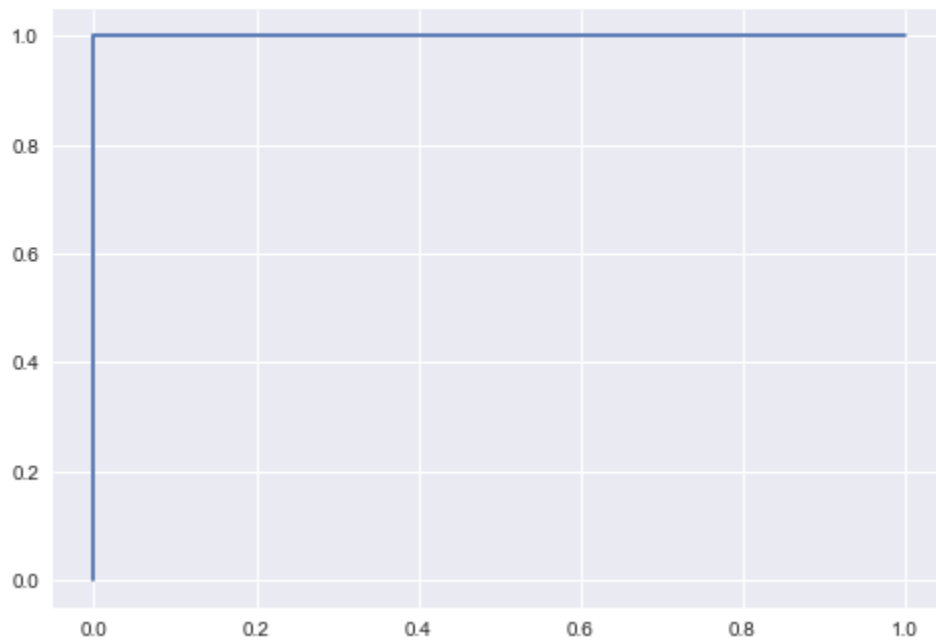


Fig:4.3.48

For RANDOM FOREST CLASSIFICATION:

```
#visualization for random forest(accuracy)  
plt.plot(fpr3,tpr3)
```

```
[<matplotlib.lines.Line2D at 0x1a9f5b27e48>]
```

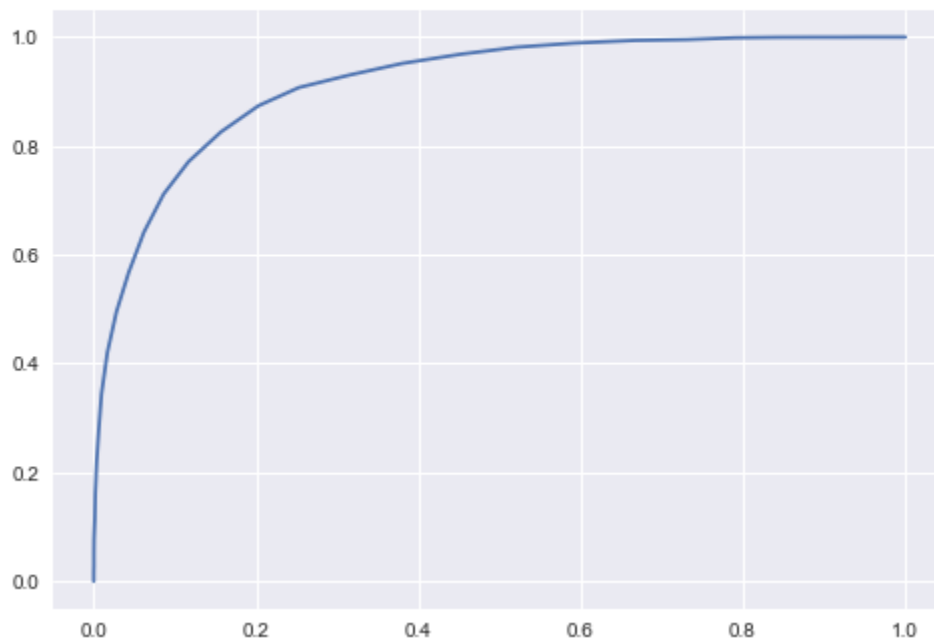


Fig:4.3.49

CHECKING THE BEST MODELS WITH UNKNOWN DATA :

After predicting the given data on all the three models , we have got the highest accuracies as 1.0 for both Naive Bayes Algorithm as well as Logistic Regression.

Now we need to give some unknown data to the models and check if they are working properly.

CHECKING LOGISTIC REGRESSION MODEL WITH UNKNOWN DATA:

```
# CHECKING THE LOGISTIC REGRESSION MODEL WITH UNKNOWN DATA
# positive review
value1=(["the movie is wonderful"])
final1=tfidf.transform(value1)
reg.predict(final1)

array(['positive'], dtype=object)

# negative review
value2=(["it is a bit too boring"])
final2=tfidf.transform(value2)
reg.predict(final2)

array(['negative'], dtype=object)
```

Fig:4.3.50

Here, we have given some random unknown new data to the model and checked if the model was classifying and predicting the correct reviews.

We passed the data into an array and have applied the tfidf vectorization. Next , we stored it in a new variable and made predictions on it.

CHECKING NAIVE BAYES MODEL WITH UNKNOWN DATA:

```
# CHECKING THE NAIVE BAYES MODEL WITH UNKNOWN DATA
# positive review
value2=(["great story"])
final2=tfidf.transform(value2)
model_BernNB.predict(final2)

array(['positive'], dtype='<U8')
```

```
# negative review
value1=(["the movie is pathetic"])
final1=tfidf.transform(value1)
model_BernNB.predict(final1)

array(['negative'], dtype='<U8')
```

Fig:4.3.51

Here, we have given some random unknown new data to the model and checked if the model was classifying and predicting the correct reviews.

We passed the data into an array and have applied the tfidf vectorization. Next , we stored it in a new variable and made predictions on it.

CONCLUSION:

From the above results, we can infer that for our problem statement, the Logistic Regression Model and Naive Bayes Model gave higher accuracy than Random Forest Classification. Apart from this, we can see that, in our auc-roc curve, we have obtained only two curves. This is because, both naive bayes and logistic regression have given the same accuracy as 1.0. Therefore, any of these two can be used for best results. The next best model is the Random Forest Classifier. Since we already got the maximum accuracy as 1.0 for logistic regression and naive bayes we do not have to perform any hyper parameter tuning (to increase the performance of the model) to the model. Heat maps are computed to get a clear understanding of the data set (which parameter has most abundant effect on the study case). The general order of performance for the model was LogisticRegression ~ NaiveBayes > RandomForestClassifier.

REFERENCES:

1.DATASET:

<https://drive.google.com/drive/folders/1vZgoh1o5xT6vehsRdfAQEX4nA79yoSVe?usp=sharing>

2.<https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/>

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

4.https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html?highlight=naive%20bayes#sklearn.naive_bayes.BernoulliNB

5. <https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html?highlight=random%20forest%20classifier#sklearn.ensemble.RandomForestClassifier>