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
CHAKILAM VARUN
221710309012

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 ofTechnology" 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   CHAKILAM VARUN

DATE: 221710309012

UNIVERSITY)

GITAM (DEEMED TO BE

Hyderabad-502329, India

Dated:

CERTIFICATE

This is to certify that the Industrial Training Report entitled "MOVIE REVIEWS CLASSIFICATION" is being submitted by CHAKILAM VARUN (221710309012) 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.

ACKNOWLEDGEMENT

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

        I would like to thank respected **Dr. N. Siva Prasad**, Pro Vice Chancellor, GITAM Hyderabad and **Prof.N.Seeta Ramaiah**, Principal, GITAM Hyderabad

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

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

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

<div align="right">

CHAKILAM VARUN
221710309012

</div>

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 the 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.

As human's opinion help in enhance product's efficiency, and since the success or the failure of a movie depends on its reviews, there is an increase in the demand and need to build a good sentiment analysis model that classifies movie reviews. In this research, tokenization is performed to transfer the input string into a word vector. The model is evaluated on a real world dataset. We use three algorithms to evaluate our data set. The results show that Logistic Regression outperforms the other classifiers. Later, hyper parameter tuning is performed on the Logistic Regression method to improve the accuracy.

Table of Contents:
LIST OF FIGURESIX

LIST OF FIGURES:

CHAPTER 1
MACHINE LEARNING

1.1 INTRODUCTION:

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

1.2 IMPORTANCE OF MACHINE LEARNING:

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

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

With the constant evolution of the field, there has been a subsequent rise in the uses, demands, and importance of machine learning. Big data has become quite a buzzword in the last few years; that's in part due to increased sophistication of machine learning, which

helps analyze those big chunks of big data. Machine learning has also changed the way data extraction, and interpretation is done by involving automatic sets of generic methods that have replaced traditional statistical techniques.

The process flow depicted here represents how machine learning works
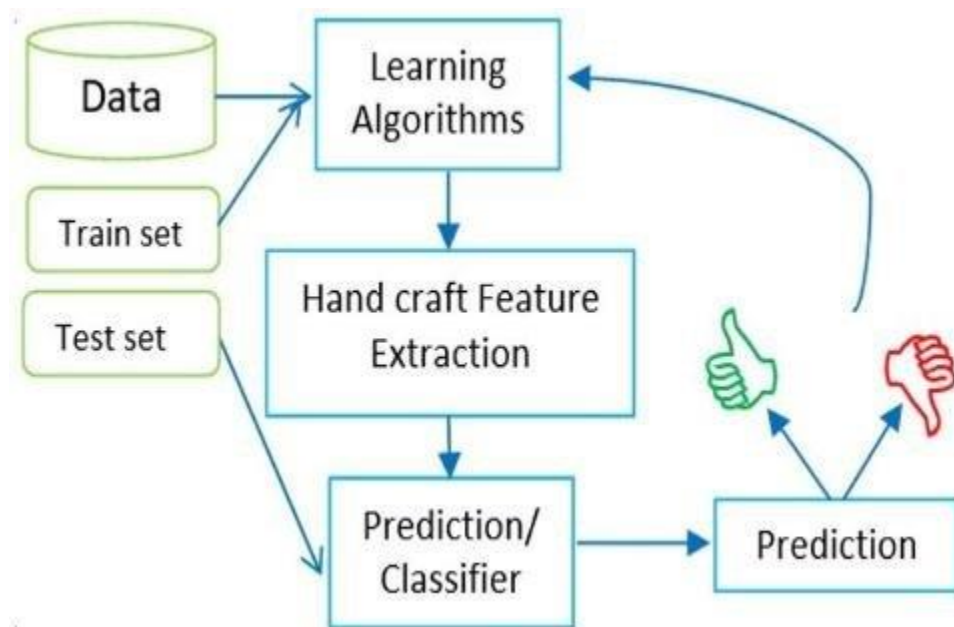
Figure 1 : The Process Flow

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.
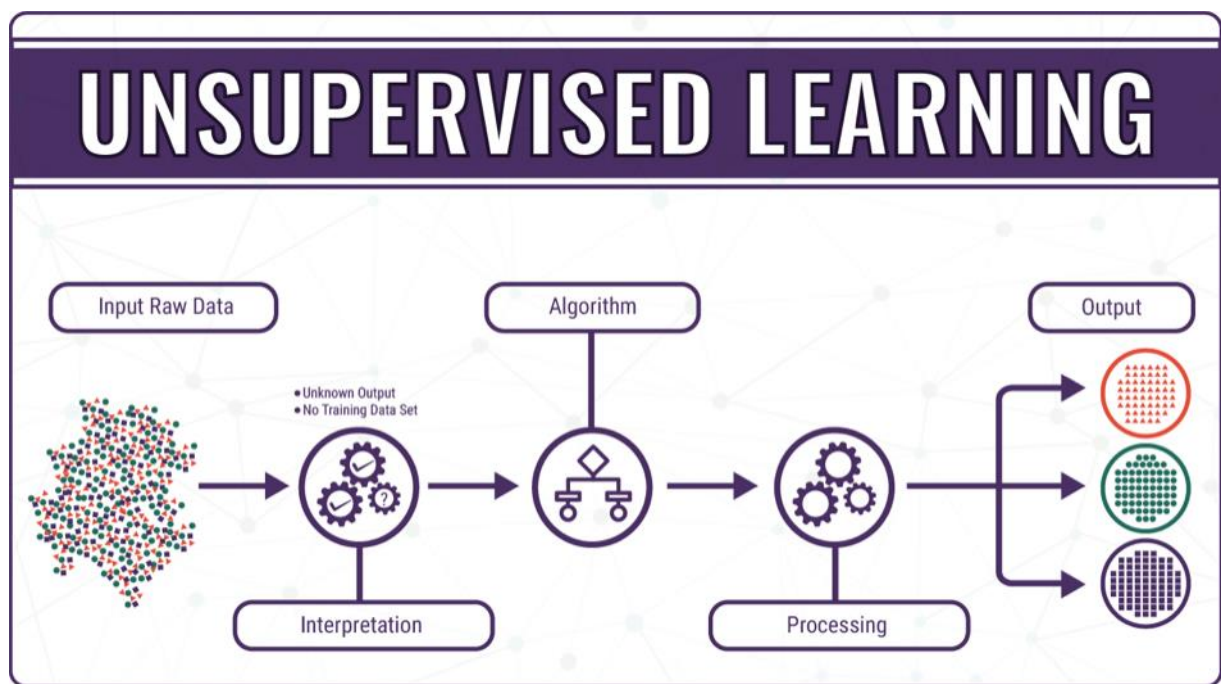
Figure 2 : Unsupervised Learning

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.
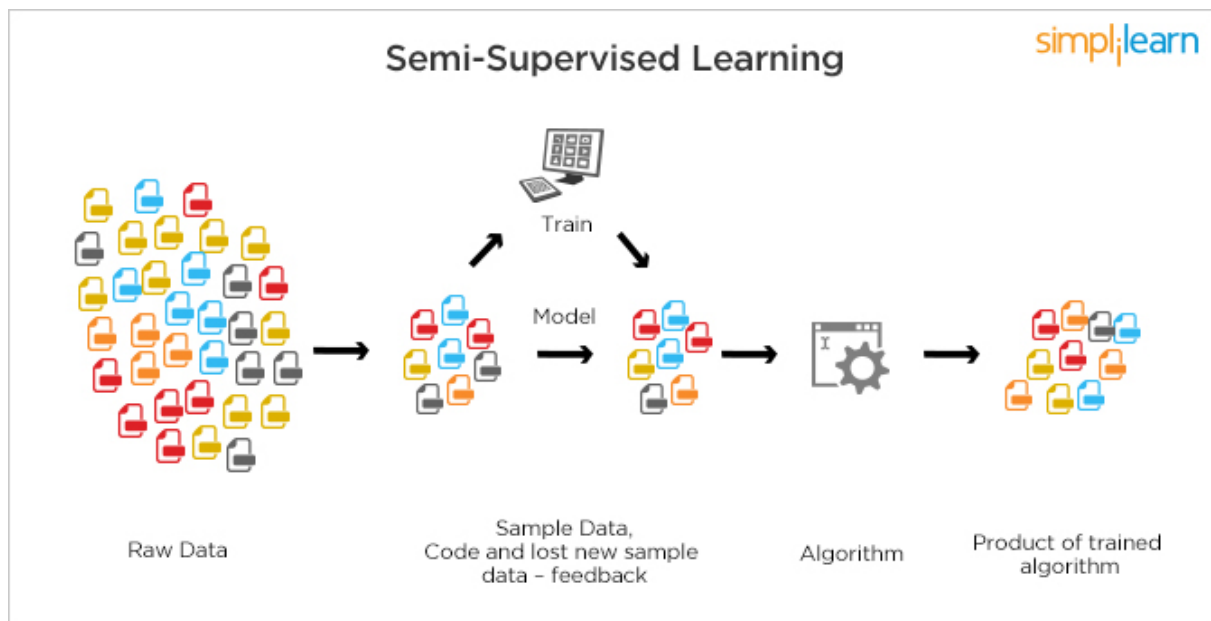
Figure 3 : Semi Supervised Learning

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 discovered previously unknown patterns and knowledge, machine learning reproduces known patterns and knowledge—and further automatically applies that information to data, decision-making, and actions.

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

CHAPTER 2
PYTHON

Basic programming language used for machine learning is : PYTHON

2.1 INTRODUCTION TO PYTHON:

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

2.2 HISTORY OF PYTHON:

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

2.3 FEATURES OF PYTHON:

- Easy-to-learn: Python has few keywords, simple structure, and a clearly defined syntax, This allows the student to pick up the language quickly.
- Easy-to-read: Python code is more clearly defined and visible to the eyes

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

2.4 HOW TO SETUP PYTHON:

- Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.
- The most up-to-date and current source code, binaries, documentation, news, etc., is available on the official website of Python.

2.4.1 Installation(using python IDLE):

- Installing python is generally easy, and nowadays many Linux and Mac OS distributions include a recent python.
- Download python from www.python.org
- When the download is completed, double click the file and follow the instructions to install it.
- When python is installed, a program called IDLE is also installed along with it. It provides a graphical user interface to work with python.
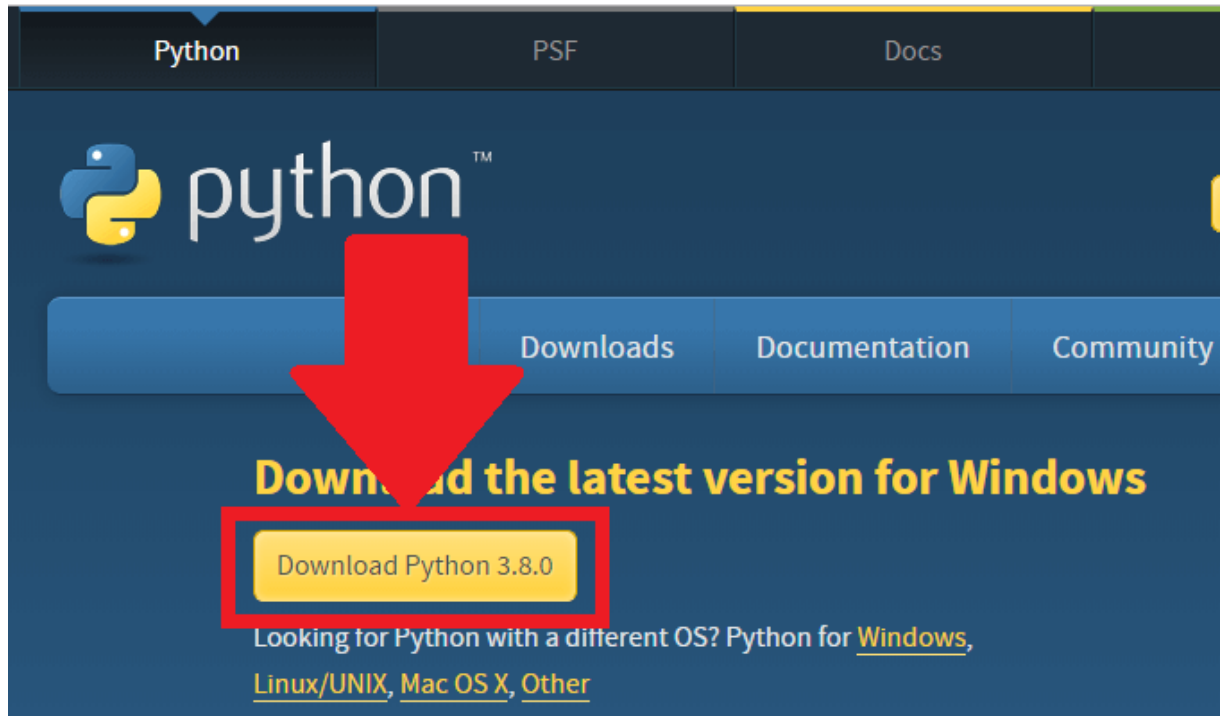
Figure 4 : Python download

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 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)
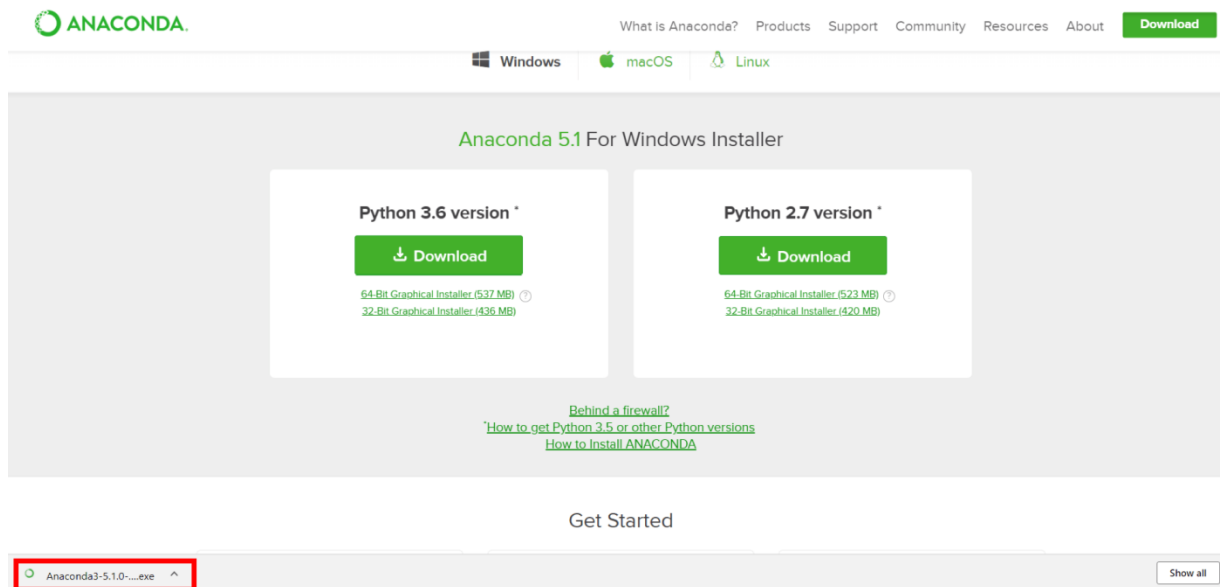
Figure 5 : Anaconda download
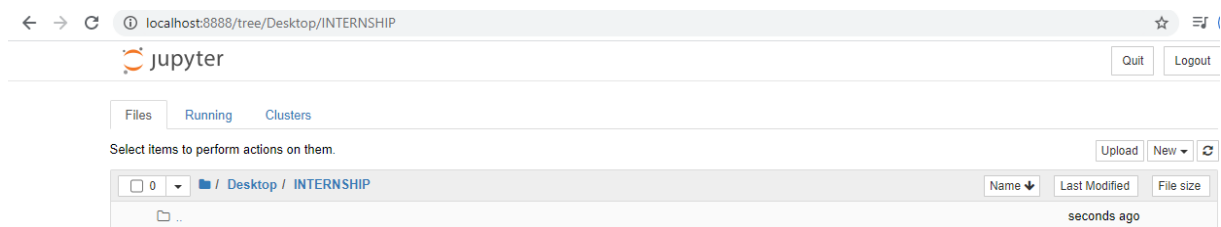
Figure 6 : Jupyter notebook

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 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 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.

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

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

12.sklearn.model.selection.GridSearchCV

 Algorithms used:

1.NAIVE BAYES ALGORITHM

2.LOGISTIC REGRESSION ALGORITHM

3.RANDOM FOREST CLASSIFIER ALGORITHM

3.1 PROBLEM STATEMENT:

To classify a given review into a positive or a negative review using sentiment analysis in Machine Learning

3.2 DATA SET:

 The given data set consists of the following parameters:

A-Review

B-Sentiment

3.3 OBJECTIVE OF THE CASE STUDY:

Movie review are an important way to gauge the performance of a movie.While providing a numerical/stars rating to a movie tells us about the success or failure of a movie quantitatively, a collection ofmovie reviews is what gives us a deeper qualitative insight on different aspects of the movie. Movie Review classification helps us in classifying the review into a positive or a negative review.Sentiment Analysis is a major subject in machine learning which aims to extract subjective information from the textual reviews.Using sentiment analysis, we can find the state of mind of the reviewer while providing the review and understand if the person was "happy", "sad", "angry" and so on, i.e. if they liked the movie or they hated it. We aim to utilize the relationships of the words in the review to predict the overall polarity of the review.

CHAPTER 4
MODEL BUILDING

4.1 PREPROCESSING OF THE DATA:

Preprocessing of the data actually involves the following steps:

4.1.1 GETTING THE DATASET:

We can get the data set from the database or

we can get the data from client.

4.1.2 IMPORTING THE LIBRARIES:

We have to import the libraries as per the requirement of the algorithm.

Figure 8 : Importing Libraries

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 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 access using the dataframe. Any missing value or NaN value have to be cleaned.

Reading the dataset

```
In [80]: data = pd.read_csv("IMDB dataset.csv")
         data.head()
```

Out[80]:

	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

Figure 9 : Reading the dataset

4.1.4 HANDLING MISSING VALUES:

Missing values can be handled in many ways using some inbuilt methods:

(a)dropna()

(b)fillna()

(c)interpolate()

(d)mean imputation and median imputation

Data Cleaning/ Data Pre-processing:

```
In [87]: data.isnull().sum()
```

```
Out[87]: review      0
         sentiment   0
         dtype: int64
```

Figure 10: Checking for missing values

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

4.1.5 STATISTICAL ANALYSIS:

- The number of rows and columns of the dataset can be found using data.shape()

```
In [81]: data.shape
Out[81]: (50000, 2)
```

Figure 11: Shape of the data

- Our data set contains 50000 rows and 2 columns
- The 2 columns in our dataset are review and sentiment columns.
- We need to describe the sentiment column and find the counts of the features.This can be done as follows:

```
In [84]: data['sentiment'].describe()
Out[84]: count        50000
         unique           2
         top       negative
         freq         25000
         Name: sentiment, dtype: object


In [85]: data.sentiment.value_counts()
Out[85]: negative    25000
         positive    25000
         Name: sentiment, dtype: int64
```

Figure 12: Features of sentiment column

Visualization of the target column i.e. the SENTIMENT column using CATPLOT .

```
In [86]: sns.catplot("sentiment",data=data,kind='count')

Out[86]: <seaborn.axisgrid.FacetGrid at 0x18338270>
```
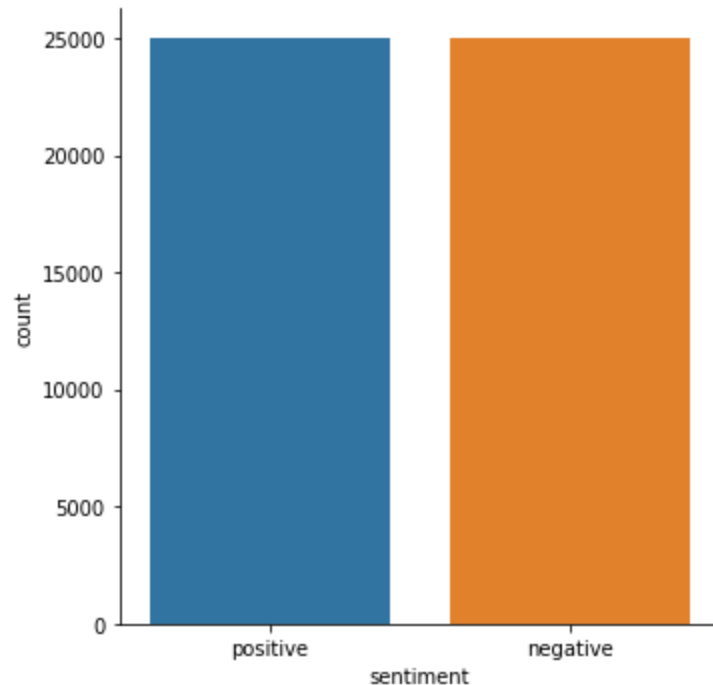

Figure 13: Visualization of the data set

4.1.6 CATEGORICAL DATA:

- Machine Learning models are based on equations, we need to replace the text by numbers. So that we can include the numbers in the equations.
- Categorical Variables are of two types: Nominal and Ordinal
- Nominal: The categories do not have any numeric ordering in between them. They don't have any ordered relationship between each of them. Examples: Male or Female, any colour
- Ordinal: The categories have a numerical ordering in between them. Example: Graduate is less than Post Graduate, Post Graduate is less

than Ph.D. customer satisfaction survey, high low medium.

```
In [66]: X=data.review
         X.head()

Out[66]: 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
```

```
In [68]: y=data.sentiment
         y.head()

Out[68]: 0    positive
         1    positive
         2    positive
         3    negative
         4    positive
         Name: sentiment, dtype: object
```

Figure 14:Categorical data

4.2 TRAINING THE MODEL:

- Splitting the data : after the preprocessing is done then the data is split into train and test sets
- 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 testset will only be available during testing the classifier.
- 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 mode has correctly learnt)
- 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%)

● First we need to identify the input and output variables and we need to separate the input set and output set

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

```
In [88]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(data.review, data.sentiment,
                                               test_size= 0.2, random_state=1 )
```

Figure 15 : importing train_test_split

After splitting train and test data the number of columns are as follows:

```
In [89]: print(X_train.shape)
         print(y_train.shape)
         print(X_test.shape)
         print(y_test.shape)

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

Figure 16:Train and Test 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 :

Count Vectorization involves counting the number of occurrences each words appears in a document.Python's Sci-kit learn library has a tool called CountVectorizer to accomplish this.

```
In [90]: ##Importing CountVectorizer
         from sklearn.feature_extraction.text import CountVectorizer

         # creating an object for CountVectorizer
         count_vect = CountVectorizer()
```

Figure 17:Importing CountVectorizer

Generating the word counts for the words in the document and getting the feature names

```
In [91]: # 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[91]: ['00',
          '000',
          '00000000000',
          '0000000000001',
          '00001',
          '00015',
          '000dm',
          '001',
          '003830',
          '0069',
          '007',
          '0079',
          '007s',
          '0080',
          '0083',
          '0093638',
          '00am',
          '00o',
          '00pm',
```

Figure 18:Generating the word counts

TFIDF VECTORIZER:

TF-IDF stands for "Term Frequency — Inverse Document Frequency".TF-IDF are word frequency scores that try to highlight words that are more interesting, e.g. frequent in a document but not across documents. The TfidfVectorizer will tokenize documents, learn the vocabulary and inverse document frequency weightings, and allow you to encode new documents.

The data present in our document is text data.Vectorisation must be performed to convert the text data,Tfidf Vectoriser is used for this.

```
In [92]: #TFIDF VECTORIZER
         from sklearn.feature_extraction.text import TfidfVectorizer
         #initialize an object for the tfidf vectorizer
         tfidf = TfidfVectorizer()
```

Figure 19:Importing tfidf Vectorizer

Applying tfidf vectorizer to test and train data to objects and generating the feature names.The input data X_train is stored in X_train_transformed, X_test is stored in X_test_transformed after applying Tf-idf vectorizer. After applying tf-idf vectoriser,our input is 'X_train_transformed' and its corresponding output is 'y_train'.It is the same for test data also.

```
In [93]: #apply the tfidf to the data(X_train)

         X_train_transformed = tfidf.fit_transform(X_train)
         X_train_transformed

Out[93]: <40000x93163 sparse matrix of type '<class 'numpy.float64'>'
                 with 5471200 stored elements in Compressed Sparse Row format>

In [94]: X_test_transformed = tfidf.transform(X_test)
         X_test_transformed

Out[94]: <10000x93163 sparse matrix of type '<class 'numpy.float64'>'
                 with 1345978 stored elements in Compressed Sparse Row format>

In [95]: #Feature names
         tfidf.get_feature_names()

Out[95]: ['00',
          '000',
          '00000000000',
          '0000000000001',
          '00001',
          '00015',
          '000dm',
          '001',
          '003830',
```

Figure 20:Vectorisation on train and test data

The position of the words in our data set can be found using Tfidf.vocabulary_

```
In [18]:  # position of the words
          tfidf.vocabulary_
          'the': 82503,
          'hilarious': 38473,
          'british': 11525,
          'comedy': 16954,
          'fish': 30690,
          'called': 12851,
          'wanda': 89606,
          'although': 3626,
          'can': 13020,
          'see': 72931,
          'why': 90722,
          'only': 58689,
          'connection': 17782,
          'find': 30513,
          'is': 42841,
          'monty': 54568,
          'python': 65803,
          'one': 58660,
          'eric': 27675,
```

<div align="center">Figure 21:Position of words</div>

4.3 EVALUATING THE CASE STUDY:

In our project,we are going to use 3 different algorithms to train and predict the data.Then,we find out the optimum algorithm that best fits our data and gives maximum accuracy.

The three algorithms which we will be using are:

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

4.3.1 NAIVE BAYES ALGORITHM:

A Naive Bayes classifier is a supervised machine learning algorithm that uses Baye's theorem which assumes that the features are statistically independent.The theorem relies on the naive assumption that input variables are independent of each other.

The equation of Baye's theorem is:

$$P(A \mid B) = \frac{P(B \mid A)P(A)}{P(B)}$$

Figure 22:Baye's Theorem equation

There are three types of Naive Bayes model under the scikit-learn library:

- **Gaussian:** It is used in classification and it assumes that features follow a normal distribution.
- **Multinomial:** It is used for discrete counts.
- **Bernoulli:** The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification with a 'bag of words' model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

We use the Bernoulli Naive Bayes model in our project.

The BernoulliNB method is imported from sklearn.naive_bayes and is assigned to a variable 'model_BernNB'

```
In [20]: # Apply the naive bayes algorithm
from sklearn.naive_bayes import BernoulliNB
#creating an object
model_BernNB = BernoulliNB()
```

Figure 23:Importing Bernoulli Naive Bayes Method

Then,we apply the method to our data and train the model using fit function.We pass X_train_transformed and y_train to train the model.

```
In [21]: # Applying the Algorithm to the data
         # ObjectName.fit(Input, Output)

         model_BernNB.fit(X_train_transformed, y_train)
Out[21]: BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

Figure 24:Fitting the model using BernoulliNb method

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 prediction on train data:

```
# Prediction on Train Data
# Syntax: objectname.predict(InputValues)
y_train_pred = model_BernNB.predict(X_train_transformed)
```

Figure 25:Prediction on train data(Naive Bayes)

To compare the actual values of y_train with the predicted values of y_train_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

```
In [23]: # 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_pred)
Out[23]: array([[18376,  1580],
                [ 2494, 17550]], dtype=int64)
```

Figure 26:Confusion matrix for train data(Naive Bayes)

The classification report and accuracy is printed as:

```
In [25]: print(classification_report(y_train,y_train_pred))
                      precision    recall  f1-score   support

            negative       0.88      0.92      0.90     19956
            positive       0.92      0.88      0.90     20044

            accuracy                           0.90     40000
           macro avg       0.90      0.90      0.90     40000
        weighted avg       0.90      0.90      0.90     40000
```

```
In [72]: accuracy_score(y_train,y_train_pred)
Out[72]: 0.89815
```

Figure 27:Classification report for train data(Naive Bayes)

To visualize the confusion matrix , we use heatmap

```
In [24]: sns.heatmap(confusion_matrix(y_train,y_train_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x14bfcf70>
```


Figure 28:Heat Map of train data(Naive Bayes)

We can conclude that the accuracy score of Naive Bayes on train data is **0.89**

For prediction on test data:

```
In [26]: # prediction on test data
         # syntax: objectname.predict(input)
         y_test_pred = model_BernNB.predict(X_test_transformed)
```

Figure 29:Prediction on test data(Naive Bayes)

To compare the actual values of y_test with the predicted values of y_test_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

```
In [27]: # compare the actual values(y_test) with predicted values(y_test_pred)

         confusion_matrix(y_test,y_test_pred)

Out[27]: array([[4431,  613],
                [ 780, 4176]], dtype=int64)
```

Figure 30:Confusion matrix for testing data(Naive Bayes)

The classification report and accuracy is printed as:

```
In [29]: accuracy_score(y_test,y_test_pred)
Out[29]: 0.8607
```

```
In [30]: print(classification_report(y_test,y_test_pred))
                  precision    recall  f1-score   support

        negative       0.85      0.88      0.86      5044
        positive       0.87      0.84      0.86      4956

        accuracy                           0.86     10000
       macro avg       0.86      0.86      0.86     10000
    weighted avg       0.86      0.86      0.86     10000
```

Figure 31:Classification report for test data(Naive Bayes)

To visualize the confusion matrix , we use heatmap

```
In [28]: sns.heatmap(confusion_matrix(y_test,y_test_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x15788790>
```
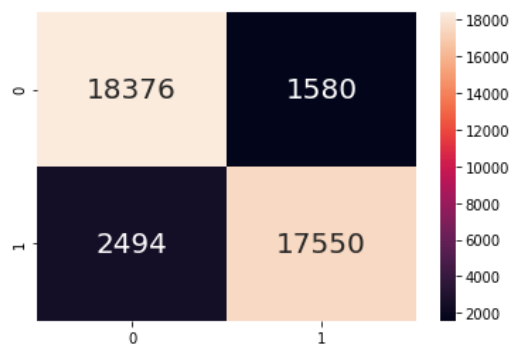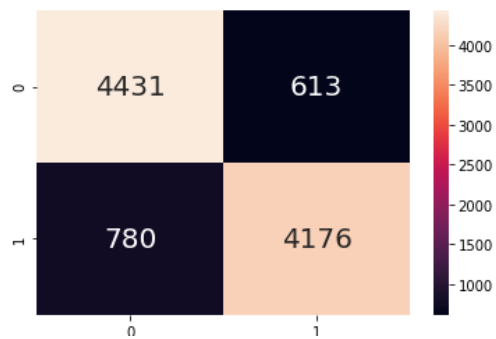

Figure 32:Heat Map of test data(Naive Bayes)

We can conclude that the accuracy score of Naive Bayes on test data is **0.86**

4.3.2 LOGISTIC REGRESSION:

Logistic regression is another technique of machine learning from the field of statistics.It is the go-to method for binary classification problems (problems with two class values).Logistic regression is used to describe data and to explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

There are three main types of logistic regression:

1. **binomial:** target variable can have only 2 possible types: "0" or "1" which may represent "win" vs "loss", "pass" vs "fail", "dead" vs "alive", etc.

2. **multinomial:** target variable can have 3 or more possible types which are not ordered(i.e. types have no quantitative significance) like "disease A" vs "disease B" vs "disease C".

3. **ordinal:** it deals with target variables with ordered categories. For example, a test score can be categorized as:"very poor", "poor", "good", "very good". Here, each category can be given a score like 0, 1, 2, 3.

   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)

```
In [31]: # 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

         C:\Users\Varun\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed
         to 'lbfgs' in 0.22. Specify a solver to silence this warning.
           FutureWarning)
```

Figure 33:Importing Logistic Regression Method and fitting the model

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 prediction on train data:

```
In [32]: #predicting on train data
         # syntax : objectname.predict(Input)
         y_train_pred =reg.predict(X_train_transformed)
         y_train_pred

Out[32]: array(['negative', 'positive', 'positive', ..., 'negative', 'negative',
                'positive'], dtype=object)
```

Figure 34:Prediction on train data(Logistic Regression)

To compare the actual values of y_train with the predicted values of y_train_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

```
In [33]: # 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_pred)
         conf

Out[33]: array([[18422,  1534],
                [ 1239, 18805]], dtype=int64)
```

Figure 35:Confusion matrix for train data(Logistic Regression)

The classification report and accuracy is printed as:

```
In [35]: from sklearn.metrics import accuracy_score
         accuracy_score(y_train,y_train_pred)

Out[35]: 0.930675

In [36]: from sklearn.metrics import classification_report
         print(classification_report(y_train,y_train_pred))

                       precision    recall  f1-score   support

             negative       0.94      0.92      0.93     19956
             positive       0.92      0.94      0.93     20044

             accuracy                           0.93     40000
            macro avg       0.93      0.93      0.93     40000
         weighted avg       0.93      0.93      0.93     40000
```

Figure 36:Classification report for train data(Logistic Regression)

To visualize the confusion matrix , we use heatmap

```
In [34]: sns.heatmap(confusion_matrix(y_train,y_train_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})

Out[34]: <matplotlib.axes._subplots.AxesSubplot at 0x161aa5d0>
```


Figure 37:Heat Map of train data(Logistic Regression)

We can conclude that the accuracy score of Logistic Regression on train data is **0.93**

For prediction on test data:

```
In [37]: #predicting on testing data
         y_test_pred =reg.predict(X_test_transformed)
         y_test_pred

Out[37]: array(['negative', 'negative', 'negative', ..., 'negative', 'positive',
                'negative'], dtype=object)
```

Figure 38:Prediction on test data(Logistic Regression)

To compare the actual values of y_test with the predicted values of y_test_pred we use a confusion matrix .

The confusion matrix is imported from sklearn.metrics package.

```
In [38]: conf = confusion_matrix(y_test,y_test_pred)
         conf

Out[38]: array([[4482,  562],
                [ 454, 4502]], dtype=int64)
```

Figure 39:Confusion matrix for test data(Logistic Regression)

The classification report and accuracy is printed as:

```
In [40]: accuracy_score(y_test,y_test_pred)
Out[40]: 0.8984

In [41]: print(classification_report(y_test,y_test_pred))
                       precision    recall  f1-score   support

             negative       0.91      0.89      0.90      5044
             positive       0.89      0.91      0.90      4956

             accuracy                           0.90     10000
            macro avg       0.90      0.90      0.90     10000
         weighted avg       0.90      0.90      0.90     10000
```

Figure 40:Classification report for test data(Logistic Regression)

To visualize the confusion matrix , we use heatmap

```
In [39]: sns.heatmap(confusion_matrix(y_test,y_test_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x15e83f50>
```
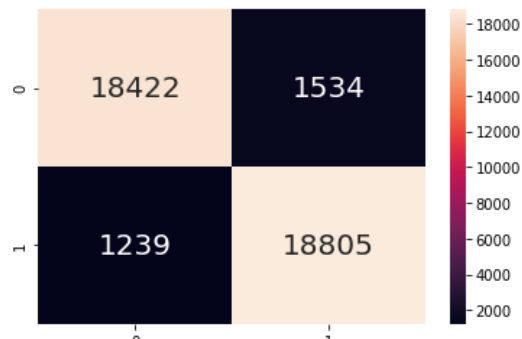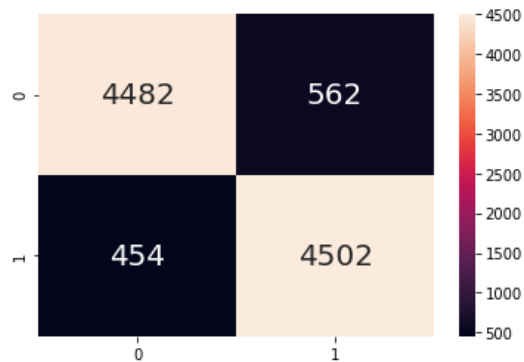

Figure 41:Heat Map of test data(Logistic Regression)

We can conclude that the accuracy score of Logistic Regression on test data is **0.89**

4.3.3 RANDOM FOREST CLASSIFICATION:

The random forest is a classification algorithm consisting of many decisions trees. It uses bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by committee is more accurate than that of any individual tree.The random forest algorithm is a supervised learning model; it uses labeled data to "learn" how to classify unlabeled .

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).

```
In [42]: #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 TFC to the dataset
         rfc.fit(X_train_transformed,y_train)

Out[42]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', 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, n_estimators=40,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

Figure 42:Importing Random Forest Classifier Method and fitting the model

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 prediction on train data:

```
In [43]: #Prediction on training data
         #Syntax: ojectname.predict(InputValues)
         y_train_pred = rfc.predict(X_train_transformed)

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

	precision	recall	f1-score	support
negative	1.00	1.00	1.00	19956
positive	1.00	1.00	1.00	20044
accuracy			1.00	40000
macro avg	1.00	1.00	1.00	40000
weighted avg	1.00	1.00	1.00	40000

Figure 43:Prediction on train data and classification report(Random Forest Classifier)

The accuracy is printed as:

```
In [45]: accuracy_score(y_train,y_train_pred)
Out[45]: 1.0
```

Figure 44:Accuracy for train data(Random Forest Classification)

To visualize the confusion matrix , we use heatmap

```
In [44]: sns.heatmap(confusion_matrix(y_train,y_train_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x15ecaff0>
```


Figure 45:Heat Map of train data(Random Forest Classifier)

We can conclude that the accuracy score of Random Forest Classifier on train data is **1.0**

For prediction on test data:

```
n [46]: #Prediction on testing data
        y_test_pred =rfc.predict(X_test_transformed)
        print(classification_report(y_test, y_test_pred))
                      precision    recall  f1-score   support

            negative       0.81      0.83      0.82      5044
            positive       0.83      0.80      0.81      4956

            accuracy                           0.82     10000
           macro avg       0.82      0.82      0.82     10000
        weighted avg       0.82      0.82      0.82     10000
```

Figure 46:Prediction on test data and classification report(Random Forest Classifier)

The accuracy is printed as:

```
In [48]: accuracy_score(y_test,y_test_pred)
Out[48]: 0.8179
```

Figure 47:Accuracy for test data(Random Forest Classification)

To visualize the confusion matrix , we use heatmap

```
In [47]: sns.heatmap(confusion_matrix(y_test,y_test_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[47]: <matplotlib.axes._subplots.AxesSubplot at 0x1540e550>
```
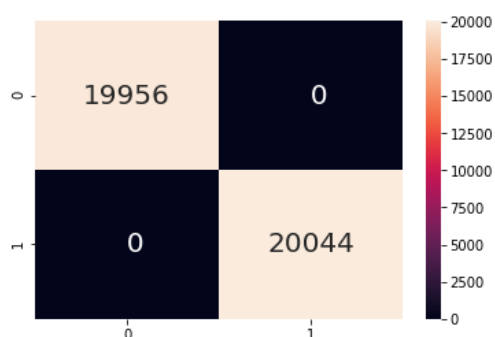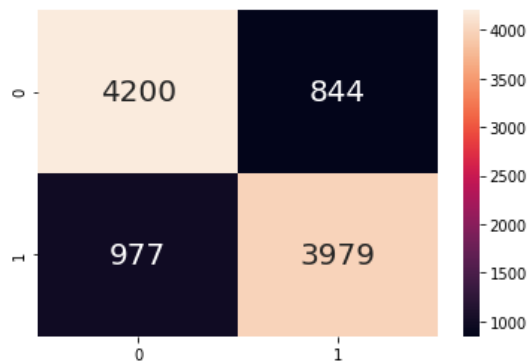

Figure 48:Heat Map of test data(Random Forest Classifier)

We can conclude that the accuracy score of Random Forest Classifier on test data is **0.81**

Comparison of accuracy scores of test data:

The accuracy scores of Naive Bayes Algorithm,Logistic Regression and Random Forest Classification on test data are 0.86,0.89 and 0.81

We can conclude that Logistic Regression algorithm has the highest accuracy value on test data and fits best for our test data

4.3.4 HYPER PARAMETER TUNING ON LOGISTIC REGRESSION

Hyper Parameter Tuning is used to increase the accuracy score of the algorithm by finding out the best parameters.We now perform Hyper Parameter tuning on Logistic Regression as it has the highest accuracy score among all the 3 methods applied on the data set.

First,we pass a list of values in a dictionary to find the optimum value for each parameter and assign it to a variable 'grid_param'

```
In [49]: #Passing list of values  in a dictionary to find the optimum value for each parameter
         grid_param = {
             'penalty': ['l1', 'l2'],
             'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
             'intercept_scaling' : [0.001, 0.01, 0.1, 1, 10, 100, 1000],
             'fit_intercept' : ['True','False']

         }
```

Figure 49:Passing a list of values as dictionary

Now,we import GridSearchCV from sklearn.model.selection.Then the initialization of GridSearch with the parameters- ModelName and the dictionary of parameters is done. Finally we apply gridsearch onto dataset.

```
In [50]: #Import the GridSearchCV
         from sklearn.model_selection import GridSearchCV

         # initialization of GridSearch with the parameters- ModelName and the dictionary of parameters
         clf = LogisticRegression()
         grid_search = GridSearchCV(estimator=clf, param_grid=grid_param)

         # applying gridsearch onto dataset
         grid_search.fit(X_train_transformed, y_train)

         C:\Users\Varun\anaconda3\lib\site-packages\sklearn\model_selection\_split.py:1978: FutureWarning: The default value of cv will
         change from 3 to 5 in version 0.22. Specify it explicitly to silence this warning.
           warnings.warn(CV_WARNING, FutureWarning)
         C:\Users\Varun\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed
         to 'lbfgs' in 0.22. Specify a solver to silence this warning.
           FutureWarning)

Out[50]: GridSearchCV(cv='warn', error_score='raise-deprecating',
```

Figure 50:Importing GridSearchCV and passing the dictionary of parameters.

We then print the best parameters as:

```
In [51]: # return the optimal parameters
         grid_search.best_params_
Out[51]: {'C': 10, 'fit_intercept': 'True', 'intercept_scaling': 0.01, 'penalty': 'l2'}
```

Figure 51:Printing the best parameters

Now,we build the model with the best parameters and fit the model with train data.

```
In [52]: #Build the model with best parameters
         #Initialized the DT Classifier
         clf = LogisticRegression(C=10, fit_intercept=True, intercept_scaling=0.01, penalty='l2')

         # We need to fit the model to the data
         clf.fit(X_train_transformed, y_train)
Out[52]: LogisticRegression(C=10, class_weight=None, dual=False, fit_intercept=True,
                            intercept_scaling=0.01, l1_ratio=None, max_iter=100,
                            multi_class='warn', n_jobs=None, penalty='l2',
                            random_state=None, solver='warn', tol=0.0001, verbose=0,
                            warm_start=False)
```

Figure 52:Building the model with best parameters

Prediction on test data:

```
In [53]: #predicting on testing data
         y_test_pred =clf.predict(X_test_transformed)
         y_test_pred
Out[53]: array(['negative', 'negative', 'negative', ..., 'positive', 'positive',
                'negative'], dtype=object)
```

Figure 53:Prediction on test data

To compare the actual values of y_test with the predicted values of y_test_pred we use a confusion matrix

```
In [54]:  conf = confusion_matrix(y_test,y_test_pred)
          conf

Out[54]:  array([[4522,  522],
                 [ 446, 4510]], dtype=int64)
```

Figure 54:Confusion Matrix of Test data(Hyper Parameter Tuning)

The classification report and accuracy scores are printed as:

```
In [56]:  accuracy_score(y_test,y_test_pred)

Out[56]:  0.9032


In [57]:  print(classification_report(y_test,y_test_pred))

                       precision    recall  f1-score   support

             negative       0.91      0.90      0.90      5044
             positive       0.90      0.91      0.90      4956

             accuracy                           0.90     10000
            macro avg       0.90      0.90      0.90     10000
         weighted avg       0.90      0.90      0.90     10000
```

Figure 55:Classification report and accuracy of test data (Hyper Parameter Tuning)

To visualize the confusion matrix , we use heatmap

```
In [55]: sns.heatmap(confusion_matrix(y_test,y_test_pred),annot=True,fmt='3.0f',annot_kws={'size':'20'})
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x156fbd70>
```
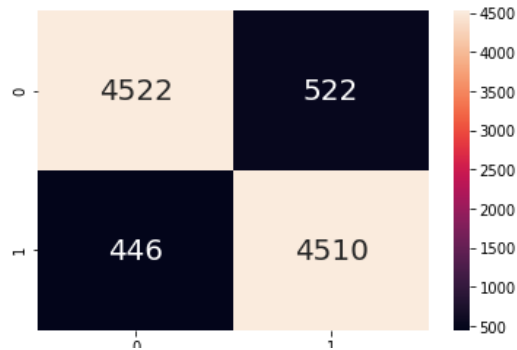

Figure 56:Heat Map of test data(Hyper Parameter Tuning)

We can clearly see that the accuracy score of Logistic Regression on test data after hyper parameter tuning has increased to 0.90 from 0.89

After building the model using the above three algorithms and hyper parameter tuning, 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

```
In [58]: from sklearn.naive_bayes import BernoulliNB
         from sklearn.linear_model import LogisticRegression
         from sklearn.ensemble import RandomForestClassifier

In [59]: model_BernNB = BernoulliNB()
         clf = LogisticRegression(C=10, fit_intercept=True, intercept_scaling=0.01, penalty='l2')
         rfc = RandomForestClassifier(n_estimators = 40)

In [60]: model_BernNB.fit(X_train_transformed,y_train)
         clf.fit(X_train_transformed,y_train)
         rfc.fit(X_train_transformed,y_train)

         C:\Users\Varun\anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed
         to 'lbfgs' in 0.22. Specify a solver to silence this warning.
           FutureWarning)

Out[60]: RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                                max_depth=None, max_features='auto', 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, n_estimators=40,
                                n_jobs=None, oob_score=False, random_state=None,
                                verbose=0, warm_start=False)
```

Figure 57:Importing packages for Roc-Auc Curve

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.

```
In [61]: pred_prob1=model_BernNB.predict_proba(X_test_transformed)
         pred_prob2=clf.predict_proba(X_test_transformed)
         pred_prob3=rfc.predict_proba(X_test_transformed)
```

```
In [62]: from sklearn.metrics import roc_curve

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

Figure 58:Calculating the predicted probabilities and importing roc auc curve

We calculate the auc scores as:

```
In [63]: from sklearn.metrics import roc_auc_score

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

         print(auc_score1,auc_score2,auc_score3)

         0.9505193332718909 1.0 0.9271553562833933
```

```
In [64]: import matplotlib.pyplot as plt
```

Figure 59:Calculating the auc scores

Now,we plot the roc-auc curve using pyplot from matplotlib

```
In [64]:  import matplotlib.pyplot as plt
          plt.style.use('seaborn')

          #plot roc curves
          plt.plot(fpr1,tpr1,linestyle='-',color='red',label='Naive Bayes')
          plt.plot(fpr2,tpr2,linestyle='-',color='green',label='Logistic Regression After Hyper Parameter Tuning')
          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()
```
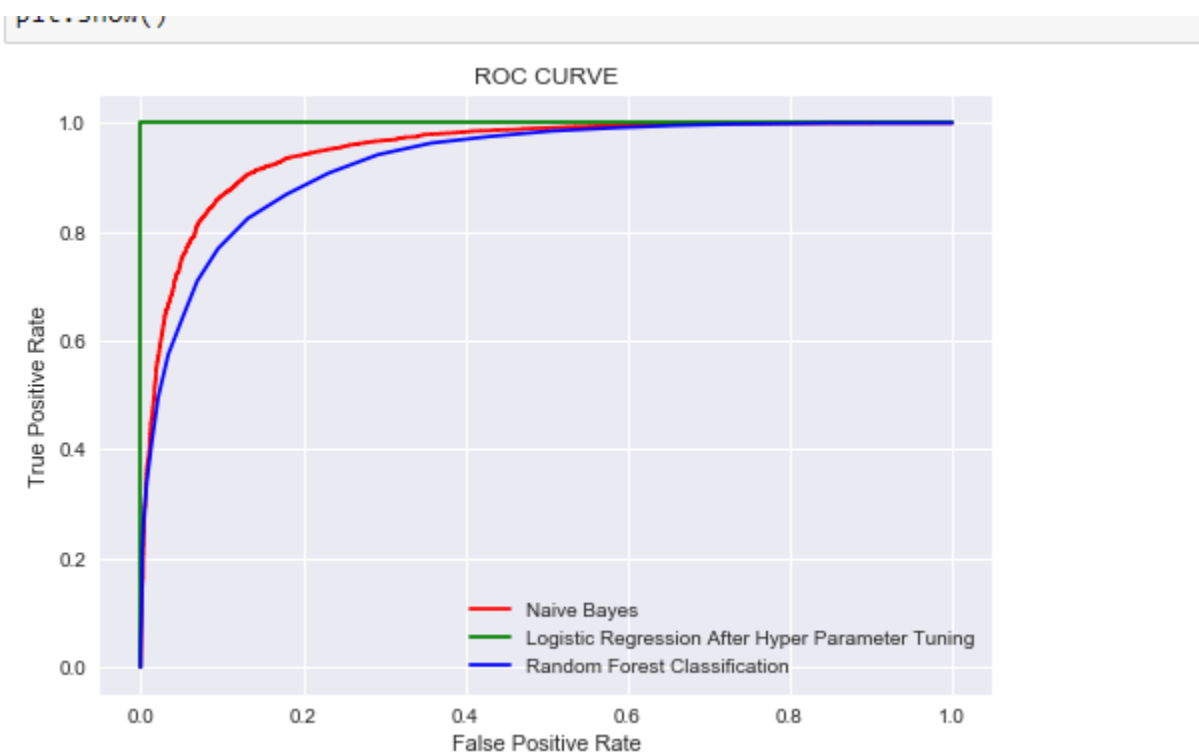
Figure 60(1):Roc Curve

Figure 60(2):Roc Curve

CONCLUSION:

From the above results,we can infer that for our problem statement,the Logistic Regression Model(with hyper parameter tuning) gave the highest accuracy than the other two models Naive bayes and random forest classification.Apart from this, we can see that the next best algorithm is the Naive Bayes. Then is the random forest classifier.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 (Hyper Parameter Tuning) > NaïveBayes > 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