# 1 INTRODUCTION

## 1.1 INTRODUCTION

The liver is a large, meaty organ that sits on the right side of the belly. Weighing about 3 pounds, the liver is reddish brown in color and feels rubbery to the feel. The liver has two large sections, called the right and the left lobes. The gallbladder sits below the liver, along with parts of the pancreas and intestines. The liver and these organs behavior together to digest, absorb, and process food. The liver's main job is to strain the blood coming from the digestive tract, before passing it to the rest of the body. The liver also detoxifies chemicals and metabolizes drugs. As it does so, the liver hides bile that ends up back in the intestines. The liver also makes proteins important for blood clotting and other functions.

Liver disease is any trouble of liver function that causes sickness. The liver is responsible for many dangerous functions within the body and should it become diseased or damaged, the loss of those functions can cause significant injury to the body. Liver disease is also referred to as hepatic disease. Liver disease is a large term that covers all the potential problems that cause the liver to fail to perform its designated functions.

## 1.2 OBJECTIVE OF THE PROJECT

This **project Liver Disease Prediction Using Machine Learning** is a machine learning application. In this project, you predict whether the patient contain a liver disease or not using python Jupyter Notebook. To predict presence of liver disease we apply some of the classification techniques.

It gives an idea of how machine learning helps in medical field and how classification techniques going to predict liver disease using liver disease data set.

## 1.3 ORGANIZTION OF THE REPORT

The first chapter deals with introduction of liver disease prediction using machine learning , motivation for developing this project, objective of the project. The second chapter deals with the system specifications required for developing the project. It includes hardware & software specifications. The third chapter gives you the Design and implementation which includes life cycle of machine learning, explanation of each cycle, flow chart of the project, visualization of the data in the dataset in form of histograms. Fourth chapter deals with the

introduction of classification techniques, explanation for each technique used in the project along with the source code and output. Finally in the fifth chapter coclusion is defined.

# 2 SYSTEM SPECIFICATION

## 2.1 SOFTWARE SPECIFICATION

- Operating system : WINDOWS
- Application software : Jupyter Notebook
- Language : python
- Data set : Liver Disease data set downloaded from Kaggle dataset

## 2.2 HARDWARE SPECIFIACTION

- Hard Disk : 32 GB
- RAM : 4 GB
- Processor : Any Pentium version

# 3 DESIGN AND IMPLEMENTATION

## 3.1 INTRODUCTION

**Machine Learning** is the field of study that gives computers the capability to learn without being explicitly programmed. ML is one of the most exciting technologies that one would have ever come across. As it is evident from the name, it gives the computer that makes it more similar to humans: **The ability to learn.**

**Machine learning** is the study of computer algorithms that improve automatically through experience. It is seen as a part of artificial intelligence. Machine learning algorithms build a model based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to do so. Machine learning algorithms are used in a wide variety of applications, such as email filtering and computer vision, where it is difficult or unfeasible to develop conventional algorithms to perform the needed tasks.

## 3.2 MACHINE LEARNING LIFE CYCLE

Machine learning life cycle is a cyclic process to build an efficient machine learning project. The main purpose of the life cycle is to find a solution to the problem or project. Machine learning life cycle involves seven major steps, which are given below:

- Data Gathering
- Data preparation
- Data wrangling
- Data Analysis
- Train model
- Test model
- Deployment

## 3.3 DATA GATHERING

Data Gathering is the first step of the machine learning life cycle. The goal of this step is to identify and obtain all data-related problems. In this step, we need to identify the different data sources, as data can be collected from various sources such as **files**, **database**, **internet**, or **mobile devices**. It is one of the most important steps of the life cycle. The quantity and

quality of the collected data will determine the efficiency of the output. The more will be the data, the more accurate will be the prediction.

This step includes the below tasks:

- **Identify various data sources**
- **Collect data**
- **Integrate the data obtained from different sources**

By performing the above task, we get a coherent set of data, also called as a **dataset**. It will be used in further steps.

Attributes from the model for liver disease prediction:

**Table 1.** Attributes from model for liver diseases diagnosis

| No | Attribute name (unit) | Range |
|---|---|---|
| 1 | Age | [4–90] |
| 2 | Gender | [Male–Female] |
| 3 | TB :Total Bilirubin (mg/dl) | [0.4–75] |
| 4 | DB: Direct Bilirubin (mg/dl) | [0.1–19.7] |
| 5 | TP: Total Protein (g/dl) | [2.7–9.6] |
| 6 | ALB: Albumin (g/dl) | [0.9–5.5] |
| 7 | A/G Ratio: Albumin and Globulin Ratio (%) | [0.3–2.8] |
| 8 | Sgpt Alamine :Aminotransferase | [10–2000] |
| 9 | Sgot Aspartate: Aminotransferase | [10–4929] |
| 10 | Alkphos: Alkaline Phosphotase | [63–2110] |

Liver disease data set used in this project is:

| age | gender | Total Bilirubin | Direct Bilirubin | Alkphos Alkaline Phosphotase | Sgpt Alamine Aminotransferase | Sgot Aspartate Aminotransferase | Total Protiens | Albumin | A/G | R |
|---|---|---|---|---|---|---|---|---|---|---|
| 65 | Female | 0.7 | 0.1 | 187 | 16 | 18 | 6.8 | 3.3 | 0.9 | 1 |
| 62 | Male | 10.9 | 5.5 | 699 | 64 | 100 | 7.5 | 3.2 | 0.74 | 1 |
| 62 | Male | 7.3 | 4.1 | 490 | 60 | 68 | 7 | 3.3 | 0.89 | 1 |
| 58 | Male | 1 | 0.4 | 182 | 14 | 20 | 6.8 | 3.4 | 1 | 1 |
| 72 | Male | 3.9 | 2 | 195 | 27 | 59 | 7.3 | 2.4 | 0.4 | 1 |

### 3.4 DATA PREPARATION

After collecting the data, we need to prepare it for further steps. Data preparation is a step where we put our data into a suitable place and prepare it to use in our machine learning training.

In this step, first, we put all data together, and then randomize the ordering of data. This step can be further divided into two processes:

- **Data exploration:**
  It is used to understand the nature of data that we have to work with. We need to understand the characteristics, format, and quality of data.A better understanding of data leads to an effective outcome. In this, we find Correlations, general trends, and outliers.
- **Data pre-processing:**
  Now the next step is pre-processing of data for its analysis.

### 3.5 DATA WRANGLING

Data wrangling is the process of cleaning and converting raw data into a useable format. It is the process of cleaning the data, selecting the variable to use, and transforming the data in a proper format to make it more suitable for analysis in the next step. It is one of the most important steps of the complete process. Cleaning of data is required to address the quality issues.

It is not necessary that data we have collected is always of our use as some of the data may not be useful. In real-world applications, collected data may have various issues, including:

- **Missing Values**
- **Duplicate data**
- **Invalid data**
- **Noise**

So, we use various filtering techniques to clean the data. It is mandatory to detect and remove the above issues because it can negatively affect the quality of the outcome.

| age | gender | Total Bilirubin | Direct Bilirubin | Alkphos Alkaline Phosphotase | Sgpt Alamine Aminotransferase | Sgot Aspartate Aminotransferase | Total Protiens | Albumin | A/G | R |
|---|---|---|---|---|---|---|---|---|---|---|
| 65 | Female | **NAN** | 0.1 | 187 | 16 | 18 | 6.8 | 3.3 | 0.9 | 1 |
| 62 | Male | 10.9 | 5.5 | 699 | 64 | **NAN** | 7.5 | 3.2 | 0.74 | 1 |
| 62 | Male | 7.3 | 4.1 | 490 | 60 | 68 | 7 | 3.3 | 0.89 | 1 |
| 58 | Male | 1 | 0.4 | **NAN** | 14 | 20 | **NAN** | 3.4 | 1 | 1 |
| 72 | Male | 3.9 | 2 | 195 | 27 | 59 | 7.3 | 2.4 | 0.4 | 1 |

### 3.6 DATA ANALYSIS

Now the cleaned and prepared data is passed on to the analysis step. This step involves:

- **Selection of analytical techniques**
- **Building models**
- **Review the result**

The aim of this step is to build a machine learning model to analyze the data using various analytical techniques and review the outcome. It starts with the determination of the type of the problems, where we select the machine learning techniques such as **Classification** i.e., **logistic regression, decision tree, random forest** etc., then build the model using prepared

data, and evaluate the model. Hence, in this step, we take the data and use machine learning algorithms to build the model.

### 3.7 TRAIN MODEL

Now the next step is to train the model, in this step we train our model to improve its performance for better outcome of the problem.

We use datasets to train the model using various machine learning algorithms. Training a model is required so that it can understand the various patterns, rules, and, features.

### 3.8 TEST MODEL

Once our machine learning model has been trained on a given dataset, then we test the model. In this step, we check for the accuracy of our model by providing a test dataset to it.

Testing the model determines the percentage accuracy of the model as per the requirement of project or problem.

### 3.9 DEPLOYMENT

The last step of machine learning life cycle is deployment, where we deploy the model in the real-world system. If the above-prepared model is producing an accurate result as per our requirement with acceptable speed, then we deploy the model in the real system. But before deploying the project, we will check whether it is improving its performance using available data or not. The deployment phase is similar to making the final report for a project.

## 3.10 FLOW CHART

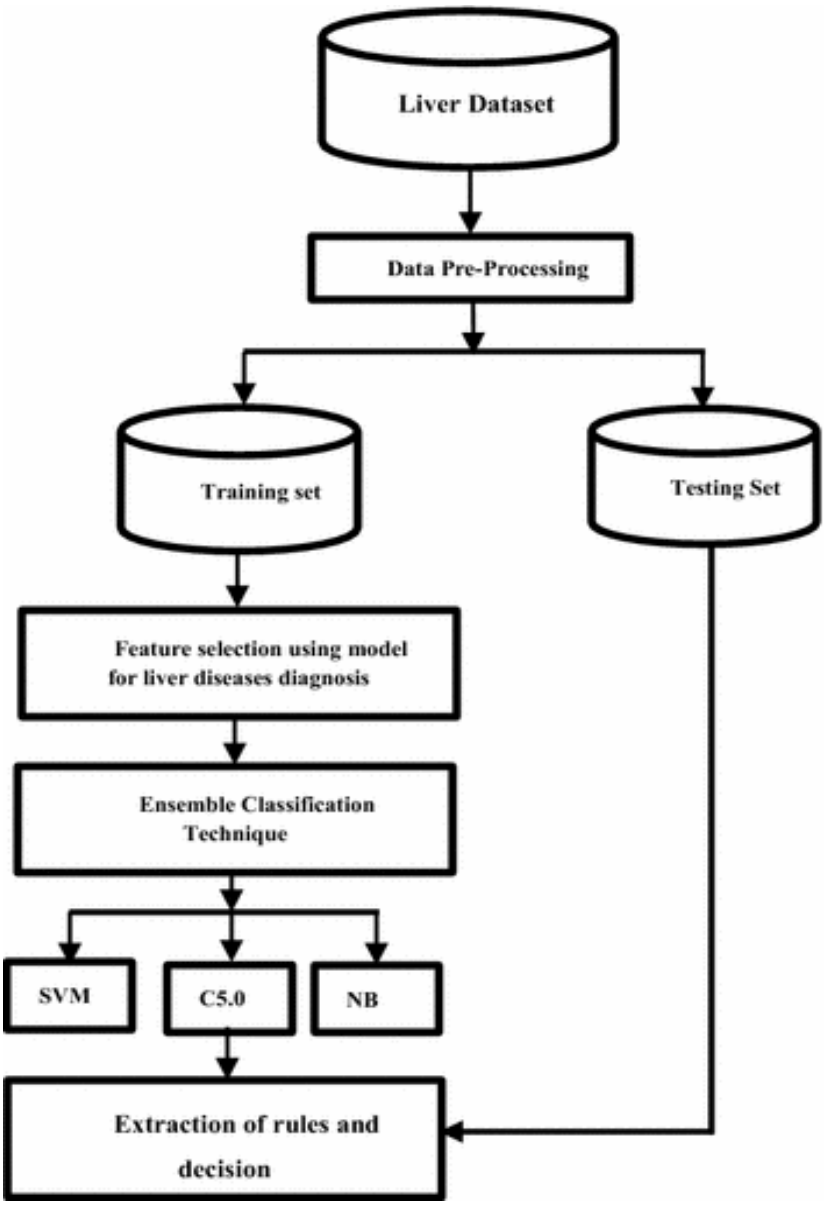
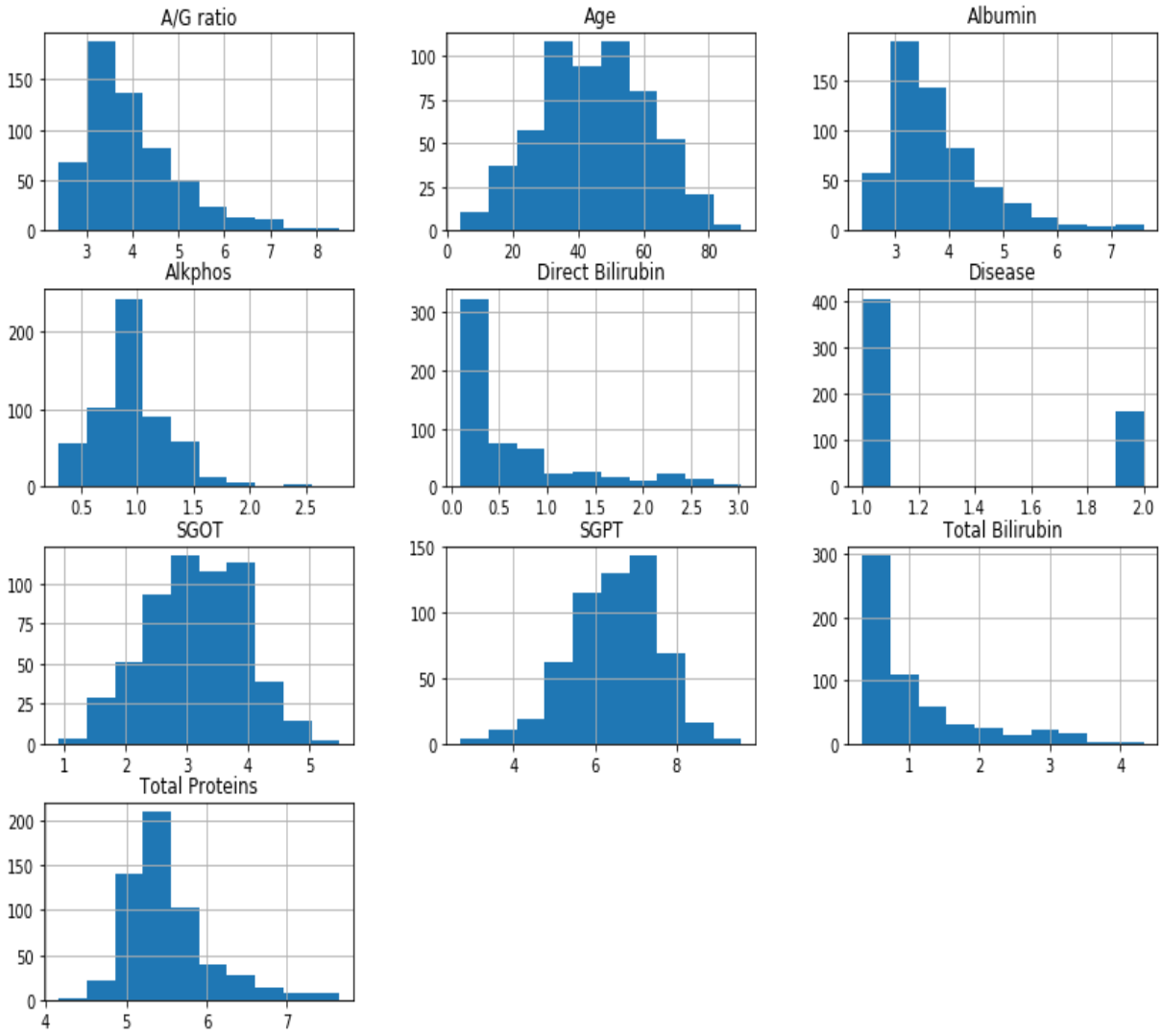



**Fig 3.10.1** Prediction of Liver Diseases Based on Machine Learning Technique

## 3.11 DATA VISUALIZATION



# 4 CLASSIFICATION TECHNIQUES

## 4.1 INTRODUCTION

Classification is defined as the process of recognition, understanding, and grouping of objects and ideas into preset categories "sub-populations." With the help of these pre-categorized training datasets, classification in machine learning programs leverage a wide range of algorithms to classify future datasets into respective and relevant categories.

Classification algorithms used in machine learning utilize input training data for the purpose of predicting the likelihood or probability that the data that follows will fall into one of the predetermined categories. One of the most common applications of classification is for filtering emails into "spam" or "non-spam", as used by today's top email service providers.

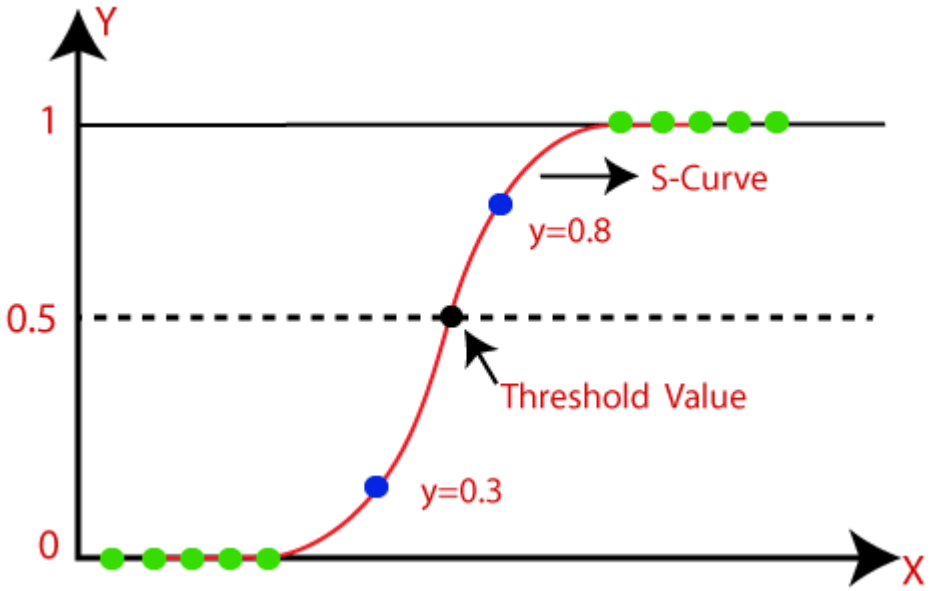
In short, classification is a form of "pattern recognition,". Here, classification algorithms applied to the training data find the same pattern.

Different classification techniques used in this project are:

- Logistic Regression
- Decision Tree
- Random Forest
- Support Vector Machine

## 4.2 LOGISTIC REGRESSION

- Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique. It is used for predicting the categorical dependent variable using a given set of independent variables.
- Logistic regression predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either Yes or No, 0 or 1, true or False, etc. but instead of giving the exact value as 0 and 1, **it gives the probabilistic values which lie between 0 and 1**.




- Logistic Regression is much similar to the Linear Regression except that how they are used. Linear Regression is used for solving Regression problems, whereas **Logistic regression is used for solving the classification problems**.
- In Logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

The Logistic regression equation can be obtained from the Linear Regression equation. The mathematical steps to get Logistic Regression equations are given below:

We know the equation of the straight line can be written as:

$$y= b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

In Logistic Regression y can be between 0 and 1 only, so for this let's divide the above equation by (1-y):

$$\frac{y}{1-y}$$ ; 0 for y= 0, and infinity for y=1

But we need range between -[infinity] to +[infinity], then take logarithm of the equation it will become:

$$log\left[\frac{y}{1-y}\right] = b_0 + b_1x_1 + b_2x_2 + b_3x_3 + \cdots + b_nx_n$$

The above equation is the final equation for Logistic Regression.

**Type of Logistic Regression:**

On the basis of the categories, Logistic Regression can be classified into three types:

- **Binomial:** In binomial Logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or Fail, etc.
- **Multinomial:** In multinomial Logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep".
- **Ordinal:** In ordinal Logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "Medium", or "High".

### 4.2.1 Python Implementation Of Logistic Regression

we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
1. #Data Pre-procesing Step
2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mp
5. import pandas as pd
6. #importing datasets
7. a=pd.read_excel(r"C:\Users\vennela\Documents\miniproject2/traindata1.xlsx
",header=0)
```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

```
  Age of the patient Gender of the patient  Total Bilirubin  \
0                  65.0                Female             0.7
1                  62.0                  Male            10.9
2                  62.0                  Male             7.3
3                  58.0                  Male             1.0
```

```
Direct Bilirubin   Alkphos Alkaline Phosphotase  \
0                   0.1                          187.0
1                   5.5                          699.0
2                   4.1                          490.0
3                   0.4                          182.0

Sgpt Alamine Aminotransferase  Sgot Aspartate Aminotransferase  \
0                               16.0                             18.0
1                               64.0                            100.0
2                               60.0                             68.0
3                               14.0                             20.0

Total Protiens   ALB Albumin  A/G Ratio Albumin and Globulin Ratio \
0                 6.8          3.3                                 0.90
1                 7.5          3.2                                 0.74
2                 7.0          3.3                                 0.89
3                 6.8          3.4                                 1.00

Result
0           1
1           1
2           1
3           1
```

The categorical values present in the gender column is replaced with female as 0 and male as 1. Below is the code for it:

**a["Gender of the patient"]=a["Gender of the patient"].replace("Female",0)**

**a["Gender of the patient"]=a["Gender of the patient"].replace("Male",1)**

Checking and removing the null values present in the dataset. Below is the code for it:

**print(a.isnull().sum())**

```
Age of the patient                    2
Gender of the patient                 902
Total Bilirubin                       648
Direct Bilirubin                      561
 Alkphos Alkaline Phosphotase         796
 Sgpt Alamine Aminotransferase        538
Sgot Aspartate Aminotransferase       462
Total Protiens                        463
 ALB Albumin                          494
A/G Ratio Albumin and Globulin Ratio  559
Result                                  0
dtype: int64
```

**a=a.dropna()**

**print(a.isnull().sum())**

```
Age of the patient                    0
Gender of the patient                 0
```

```
Total Bilirubin                          0
Direct Bilirubin                         0
 Alkphos Alkaline Phosphotase            0
 Sgpt Alamine Aminotransferase           0
Sgot Aspartate Aminotransferase          0
Total Protiens                           0
 ALB Albumin                             0
A/G Ratio Albumin and Globulin Ratio     0
Result                                   0
dtype: int64
```

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

**#Extracting Independent and dependent Variable**

**x=a.iloc[:,2:-1]**

**y=np.array(a.iloc[:,10:])**

In the above code, we have taken [2, 3, 4, 5, 6, 7, 8, 9] for x because our independent Variables are **Total Bilirubin, Direct Bilirubin, Alkphos Alkaline Phosphotase , Sgpt Ala mine Aminotransferase, Sgot Aspartate Aminotransferase, Total Protiens, ALB Albumi n A/G Ratio Albumin and Globulin Ratio** , which are at index 2, 3, 4, 5, 6, 7, 8, 9. And we have taken 10 for y variable because our dependent variable is at index 10.

Now we will split the dataset into a training set and test set. Below is the code for it:

1. **# Splitting the dataset into training and test set.**
2. **from sklearn.model_selection import train_test_split**
3. **X_train, X_test, Y_train, Y_test= train_test_split(x, y, random_state=0)**

In logistic regression, we will do feature scaling because we want accurate result of predictions. Here we will only scale the independent variable because dependent variable have only 0 and 1 values. Below is the code for it:

1. **#feature Scaling**
2. **from sklearn.preprocessing import StandardScaler**
3. **st_x= StandardScaler()**
4. **x_train= st_x.fit_transform(x_train)**
5. **x_test= st_x.transform(x_test)**

**Fitting Logistic Regression to the Training set:**

We have well prepared our dataset, and now we will train the dataset using the training set. For providing training or fitting the model to the training set, we will import the **LogisticRegression** class of the **sklearn** library.

After importing the class, we will create a classifier object and use it to fit the model to the logistic regression. Below is the code for it:

```
1. #Fitting Logistic Regression to the training set
2. from sklearn.linear_model import LogisticRegression
3. classifier= LogisticRegression(random_state=0)
4. classifier.fit(x_train, y_train)
```

By executing the above code, we will get the below output:

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

**Predicting the Test Result**

Our model is well trained on the training set, so we will now predict the result by using test set data. Below is the code for it:

```
1. #Predicting the test set result
2. Predicted_y= classifier.predict(x_test)
```

In the above code, we have created a predicted_y vector to predict the test set result.

By executing the above code, a new vector (prediction_y) will be created under the variable explorer option. It can be seen as:

```
[1 1 1 ... 1 1 1]
```

**Test Accuracy of the result**

Now we will create the confusion matrix here to check the accuracy of the classification. To create it, we need to import the **confusion_matrix** function of the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **Y_test** (the actual values) and **predicted_y** (the targeted value return by the classifier). Below is the code for it:

```
1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. cm = confusion_matrix(Y_test, predicted_y)
4. print(cm)
```

```
[[4595  231]
[1667  297]]
```

We will also create accuracy score and classification report to analyse classification. To create it, we need to import the **accuracy_score** function and **classification_report** function from the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **Y_test** (the actual values) and **predicted_y** (the targeted value return by the classifier). Below is the code for it:

- Accuracy is the percent of correct classifications and can be defined as:

$$\text{Accuracy} = (TP + TN/(TP + FP + TN + FN)) \times 100$$

- Sensitivity is the rate of true positive and can be defined as:

$$\text{Precision} = (TP/(TP + FN)) \times 100$$

- Specificity is the true negative rate and can be defined as:

$$\text{Recall} = (TN/(TN + FP)) \times 100$$

Where:

TP = the number of positive examples correctly classified.

FP = the number of positive examples misclassified as negative

FN = the number of negative examples misclassified as positive

TN = the number of negative examples correctly classified

```
1. #Creating the  Accuracy Score and classification report
2. from sklearn.metrics import accuracy_score ,classification_report
3. print(accuracy_score(predicted_y, Y_test))
4. print(classification_report(predicted_y, Y_test))
```

0.7204712812960236

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 0.95 | 0.73 | 0.83 | 6262 |
| 2 | 0.15 | 0.56 | 0.24 | 528 |
| accuracy | | | 0.72 | 6790 |
| macro avg | 0.55 | 0.65 | 0.53 | 6790 |
| weighted avg | 0.89 | 0.72 | 0.78 | 6790 |

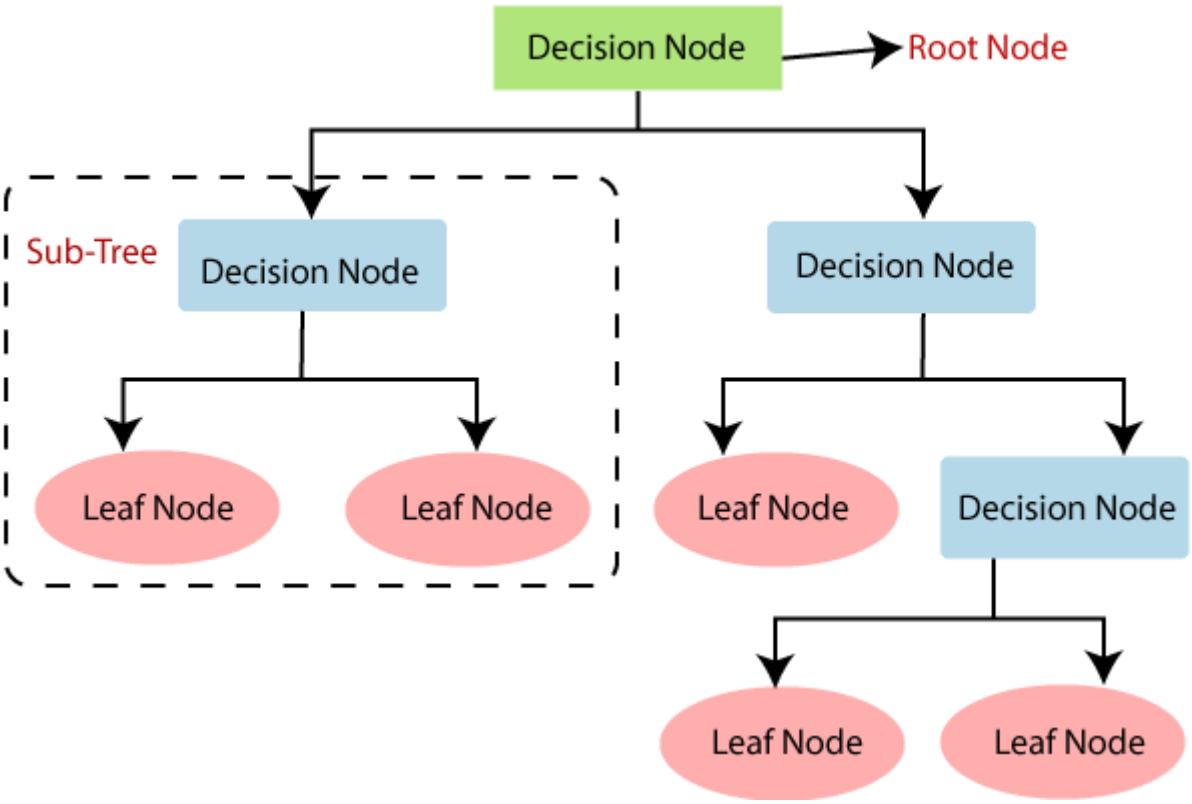
### 4.3 DECISION TREE

- Decision Tree is a **Supervised learning technique** that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where **internal nodes represent the features of a dataset, branches represent the decision rules** and **each leaf node represents the outcome.**
- In a Decision tree, there are two nodes, which are the **Decision Node** and **Leaf Node.** Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.
- In order to build a tree, we use the **CART algorithm,** which stands for **Classification and Regression Tree algorithm.**
- A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into subtrees.

- Below diagram explains the general structure of a decision tree:

**Decision Tree Terminologies:**

- **Root Node:** Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- **Leaf Node:** Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- **Splitting:** Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- **Branch/Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Pruning is the process of removing the unwanted branches from the tree.
- **Parent/Child node:** The root node of the tree is called the parent node, and other nodes are called the child nodes.



**Attribute Selection Measures:**

While implementing a Decision tree, the main issue arises that how to select the best attribute for the root node and for sub-nodes. So, to solve such problems there is a technique which is called as **Attribute selection measure or ASM.** By this measurement, we can easily select the best attribute for the nodes of the tree. There are two popular techniques for ASM, which are:

- **Information Gain**
- **Gini Index**

**1.Information Gain:**

- Information gain is the measurement of changes in entropy after the segmentation of a dataset based on an attribute.
- It calculates how much information a feature provides us about a class.
- According to the value of information gain, we split the node and build the decision tree.
- A decision tree algorithm always tries to maximize the value of information gain, and a node/attribute having the highest information gain is split first. It can be calculated using the below formula:

**Information Gain= Entropy(S)- [(Weighted Avg) *Entropy(each feature)**

**Entropy:** Entropy is a metric to measure the impurity in a given attribute. It specifies randomness in data. Entropy can be calculated as:

**Entropy(s)= -P(yes)log2 P(yes)- P(no) log2 P(no)**

**Where,**

- S= Total number of samples
- P(yes)= probability of yes
- P(no)= probability of no

**2. Gini Index:**

- Gini index is a measure of impurity or purity used while creating a decision tree in the **CART**(Classification and Regression Tree) algorithm.
- An attribute with the low Gini index should be preferred as compared to the high Gini index.
- It only creates binary splits, and the CART algorithm uses the Gini index to create binary splits.
- Gini index can be calculated using the below formula:

**Gini Index= 1- $\sum_j P_j^2$**

### 4.3.1 Python Implementation Of Decision Tree:

we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
#Data Pre-procesing Step
# importing libraries
import numpy as nm
import matplotlib.pyplot as mp
import pandas as pd
#importing datasets
a=pd.read_excel(r"C:\Users\vennela\Documents\miniproject2/traindata1.xlsx
",header=0)
```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

```
  Age of the patient Gender of the patient  Total Bilirubin  \
0                  65.0                Female             0.7
1                  62.0                  Male            10.9
2                  62.0                  Male             7.3
3                  58.0                  Male             1.0


Direct Bilirubin   Alkphos Alkaline Phosphotase  \
0                  0.1                          187.0
1                  5.5                          699.0
2                  4.1                          490.0
3                  0.4                          182.0

Sgpt Alamine Aminotransferase  Sgot Aspartate Aminotransferase  \
```

```
0                                  16.0                                  18.0
1                                  64.0                                 100.0
2                                  60.0                                  68.0
3                                  14.0                                  20.0

Total Protiens   ALB Albumin  A/G Ratio Albumin and Globulin Ratio \
0                 6.8           3.3                                   0.90
1                 7.5           3.2                                   0.74
2                 7.0           3.3                                   0.89
3                 6.8           3.4                                   1.00

Result
0           1
1           1
2           1
3           1
```

The categorical values present in the gender column is replaced with female as 0 and male as 1. Below is the code for it:

**a["Gender of the patient"]=a["Gender of the patient"].replace("Female",0)**

**a["Gender of the patient"]=a["Gender of the patient"].replace("Male",1)**

Checking and removing the null values present in the dataset. Below is the code for it:

**print(a.isnull().sum())**

```
Age of the patient                          2
Gender of the patient                     902
Total Bilirubin                           648
Direct Bilirubin                          561
 Alkphos Alkaline Phosphotase             796
 Sgpt Alamine Aminotransferase            538
Sgot Aspartate Aminotransferase           462
Total Protiens                            463
 ALB Albumin                              494
A/G Ratio Albumin and Globulin Ratio      559
Result                                      0
dtype: int64
```

**a=a.dropna()**

**print(a.isnull().sum())**

```
Age of the patient                        0
Gender of the patient                     0
Total Bilirubin                           0
Direct Bilirubin                          0
 Alkphos Alkaline Phosphotase             0
 Sgpt Alamine Aminotransferase            0
Sgot Aspartate Aminotransferase           0
Total Protiens                            0
 ALB Albumin                              0
```

```
A/G Ratio Albumin and Globulin Ratio    0
Result                                  0
dtype: int64
```

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

**#Extracting Independent and dependent Variable**

**x=a.iloc[:,2:-1]**

**y=np.array(a.iloc[:,10:])**

In the above code, we have taken [2, 3, 4, 5, 6, 7, 8, 9] for x because our independent Variables are **Total Bilirubin, Direct Bilirubin, Alkphos Alkaline Phosphotase , Sgpt Ala mine Aminotransferase, Sgot Aspartate Aminotransferase, Total Protiens, ALB Albumi n A/G Ratio Albumin and Globulin Ratio** , which are at index 2, 3, 4, 5, 6, 7, 8, 9. And we have taken 10 for y variable because our dependent variable is at index 10.

Now we will split the dataset into a training set and test set. Below is the code for it:

1. **# Splitting the dataset into training and test set.**
2. **from sklearn.model_selection import train_test_split**
4. **X_train, X_test, Y_train, Y_test= train_test_split(x, y, random_state=0)**

**Fitting a Decision-Tree algorithm to the Training set**

Now we will fit the model to the training set. For this, we will import the **DecisionTreeClassifier** class from **sklearn.tree** library. Below is the code for it:

1. **#Fitting Decision Tree classifier to the training set**
2. **from sklearn import tree**
3. **From sklearn.tree import DecisionTreeClassifier**
4. **classifier= DecisionTreeClassifier()**
5. **clf_d.fit(x_train, y_train)**

Below is the output for this:

DecisionTreeClassifier(class_weight=None,criterion='entropy',max_depth=None, max_features=None,max_leaf_nodes=None,min_impurity_decrease=0.0,min_impurity_split =None,min_samples_leaf=1,min_samples_split=2,min_weight_fraction_leaf=0.0,presort=Fal se,random_state=0, splitter='best')

**Predicting the test result**

Now we will predict the test set result. We will create a new prediction vector **y_predict.** Below is the code for it:

1. **#Predicting the test set result**
2. **y_predict=clf_d.predict(X_test)**

In the above code, we have created a y_predict vector to predict the test set result.

By executing the above code, a new vector (y_predict) will be created under the variable explorer option. It can be seen as:

**[1 1 1 ... 1 1 1]**

**Test accuracy of the result:**

If we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

- Accuracy is the percent of correct classifications and can be defined as:

$$\text{Accuracy} = (TP + TN/( TP + FP + TN + FN)) \times 100$$

- Sensitivity is the rate of true positive and can be defined as:

$$\text{Precision} = (TP/(TP + FN ))\times 100$$

- Specificity is the true negative rate and can be defined as:

$$\text{Recall} = (TN/(TN + FP)) \times 100$$

Where:

TP = the number of positive examples correctly classified.

FP = the number of positive examples misclassified as negative

FN = the number of negative examples misclassified as positive

TN = the number of negative examples correctly classified

```
1. #Creating the Confusion matrix
2. from sklearn.metrics import confusion_matrix
3. print(confusion_matrix(Y_test,y_predict))
```

output:

```
[[4824    2]
[   1 1963]]
```

We will also create accuracy score and classification report to analyse classification. To create it, we need to import the **accuracy_score** function and **classification_report** function from the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **Y_test** (the actual values) and **y_predict** (the targeted value return by the classifier). Below is the code for it:

```
1. #Creating the  Accuracy Score and classification report
2. from sklearn.metrics import accuracy_score ,classification_report
3. print(accuracy_score(y_predict, Y_test))
4. print(classification_report(y_predict, Y_test))
```

Output for the above code is as follows:

0.9995581737849779

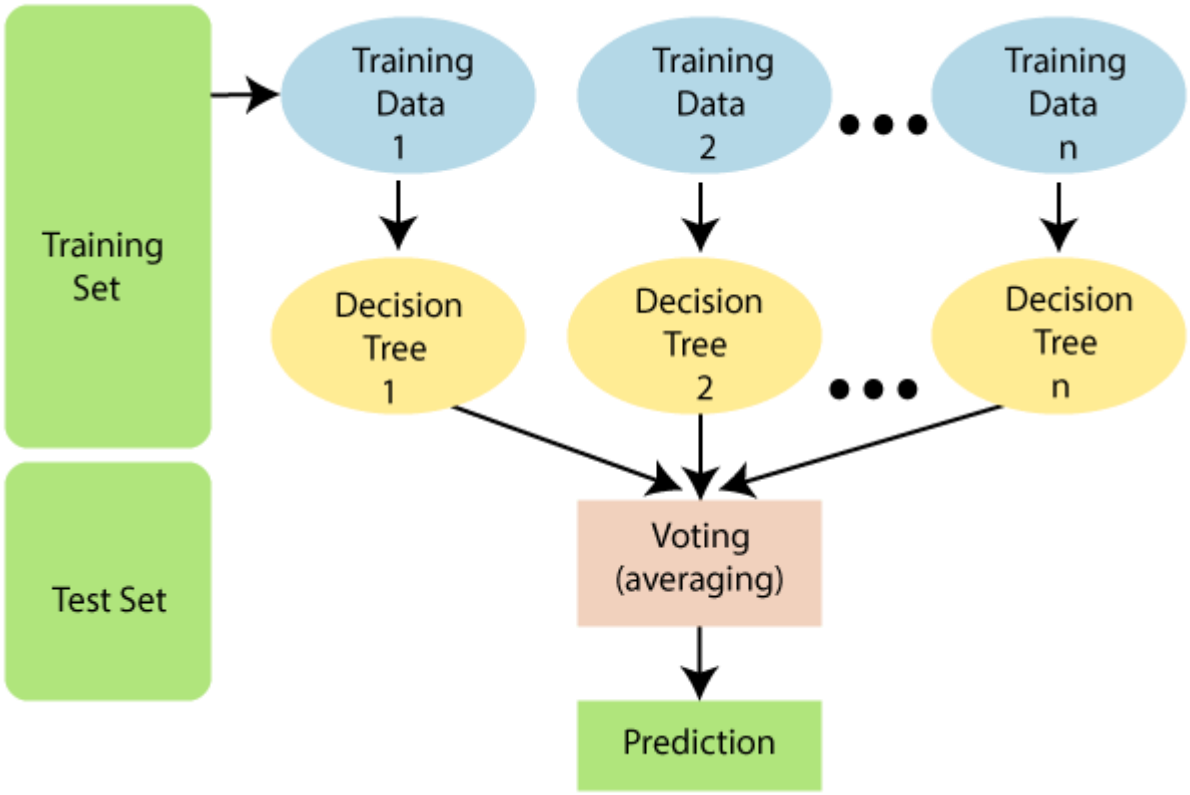
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 4825 |
| 2 | 1.00 | 1.00 | 1.00 | 1965 |
| accuracy | | | 1.00 | 6790 |

| | | | | |
|---|---|---|---|---|
| macro avg | 1.00 | 1.00 | 1.00 | 6790 |
| weighted avg | 1.00 | 1.00 | 1.00 | 6790 |

### 4.4 RANDOM FOREST

- Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*
- As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.
- **The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.**

The below diagram explains the working of the Random Forest algorithm:



**Assumptions for Random Forest:**

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

### 4.4.1 Python Implementation Of Random Forest

we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
1. #Data Pre-procesing Step
2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mp
5. import pandas as pd
6. #importing datasets
7. a=pd.read_excel(r"C:\Users\vennela\Documents\miniproject2/traindata1.xlsx
   ",header=0)
```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

```
  Age of the patient Gender of the patient  Total Bilirubin  \
0                  65.0                Female              0.7
1                  62.0                  Male             10.9
2                  62.0                  Male              7.3
3                  58.0                  Male              1.0


Direct Bilirubin   Alkphos Alkaline Phosphotase  \
0                  0.1                          187.0
1                  5.5                          699.0
2                  4.1                          490.0
3                  0.4                          182.0

Sgpt Alamine Aminotransferase  Sgot Aspartate Aminotransferase  \
0                              16.0                              18.0
```

```
1                                  64.0                                100.0
2                                  60.0                                 68.0
3                                  14.0                                 20.0

Total Protiens   ALB Albumin  A/G Ratio Albumin and Globulin Ratio \
0                 6.8           3.3                                  0.90
1                 7.5           3.2                                  0.74
2                 7.0           3.3                                  0.89
3                 6.8           3.4                                  1.00

Result
0           1
1           1
2           1
3           1
```

The categorical values present in the gender column is replaced with female as 0 and male as 1. Below is the code for it:

**a["Gender of the patient"]=a["Gender of the patient"].replace("Female",0)**

**a["Gender of the patient"]=a["Gender of the patient"].replace("Male",1)**

Checking and removing the null values present in the dataset. Below is the code for it:

**print(a.isnull().sum())**

```
Age of the patient                    2
Gender of the patient                 902
Total Bilirubin                       648
Direct Bilirubin                      561
 Alkphos Alkaline Phosphotase         796
 Sgpt Alamine Aminotransferase        538
Sgot Aspartate Aminotransferase       462
Total Protiens                        463
 ALB Albumin                          494
A/G Ratio Albumin and Globulin Ratio  559
Result                                  0
dtype: int64
```

**a=a.dropna()**

**print(a.isnull().sum())**

```
Age of the patient                    0
Gender of the patient                 0
Total Bilirubin                       0
Direct Bilirubin                      0
 Alkphos Alkaline Phosphotase         0
 Sgpt Alamine Aminotransferase        0
Sgot Aspartate Aminotransferase       0
Total Protiens                        0
 ALB Albumin                          0
A/G Ratio Albumin and Globulin Ratio  0
```

```
Result                                  0
dtype: int64
```

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

**#Extracting Independent and dependent Variable**

**x=a.iloc[:,2:-1]**

**y=np.array(a.iloc[:,10:])**

In the above code, we have taken [2, 3, 4, 5, 6, 7, 8, 9] for x because our independent Variables are **Total Bilirubin, Direct Bilirubin, Alkphos Alkaline Phosphotase , Sgpt Ala mine Aminotransferase, Sgot Aspartate Aminotransferase, Total Protiens, ALB Albumi n A/G Ratio Albumin and Globulin Ratio** , which are at index 2, 3, 4, 5, 6, 7, 8, 9. And we have taken 10 for y variable because our dependent variable is at index 10.

Now we will split the dataset into a training set and test set. Below is the code for it:

1. **# Splitting the dataset into training and test set.**
2. **from sklearn.model_selection import train_test_split**
3. **X_train, X_test, Y_train, Y_test= train_test_split(x, y, random_state=0)**

**Fitting the Random Forest algorithm to the training set:**

Now we will fit the Random forest algorithm to the training set. To fit it, we will import the **RandomForestClassifier** class from the **sklearn.ensemble** library. The code is given below:

1. **#Fitting Random Forest classifier to the training set**
2. **from sklearn.ensemble import RandomForestClassifier**
3. **clf_r=RandomForestClassifier(n_estimators=500)**
4. **clf_r.fit(X_train,Y_train)**

In the above code, the classifier object takes below parameter

- **n_estimators=** The required number of trees in the Random Forest. The default value is 10. We can choose any number but need to take care of the overfitting issue.

By executing the above code, we will get the below output:

RandomForestClassifier(bootstrap=True,class_weight=None,criterion='entropy',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=10,n_jobs=None,oob_score=False,random_state=None,verbose=0,warm_start=False)

**Predicting the Test Set result**

Since our model is fitted to the training set, so now we can predict the test result. For prediction, we will create a new prediction vector y_pred_r. Below is the code for it:

1. **#Predicting the test set result**
2. **y_pred_r=clf_r.predict(X_test)**

In the above code, we have created a y_pred_r vector to predict the test set result.

By executing the above code, a new vector (y_pred_r) will be created under the variable explorer option. It can be seen as:

**[1 1 1 ... 1 1 1]**

**Test accuracy of the result:**

If we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

1. **#Creating the Confusion matrix**
2. **from sklearn.metrics import confusion_matrix**
3. **print(confusion_matrix(y_pred_r,Y_test))**

output:

[[4824 2]
[ 1 1963]]

We will also create accuracy score and classification report to analyse classification. To create it, we need to import the **accuracy_score** function and **classification_report** function from the

sklearn library. The function takes two parameters, mainly **Y_test** (the actual values) and y_**predict_r** (the targeted value return by the classifier). Below is the code for it:

- Accuracy is the percent of correct classifications and can be defined as:

$$\text{Accuracy} = (TP + TN/(\ TP + FP + TN + FN)) \times 100$$

- Sensitivity is the rate of true positive and can be defined as:

$$\text{Precision} = (TP/(TP + FN\ ))\times 100$$

- Specificity is the true negative rate and can be defined as:

$$\text{Recall} = (TN/(TN + FP)) \times 100$$

Where:

TP = the number of positive examples correctly classified.

FP = the number of positive examples misclassified as negative

FN = the number of negative examples misclassified as positive

TN = the number of negative examples correctly classified

1. **#Creating the Accuracy Score and classification report**
2. **from sklearn.metrics import accuracy_score ,classification_report**
3. **print(accuracy_score(y_pred_r,Y_test))**
4. **print(classification_report(y_pred_r,Y_test))**

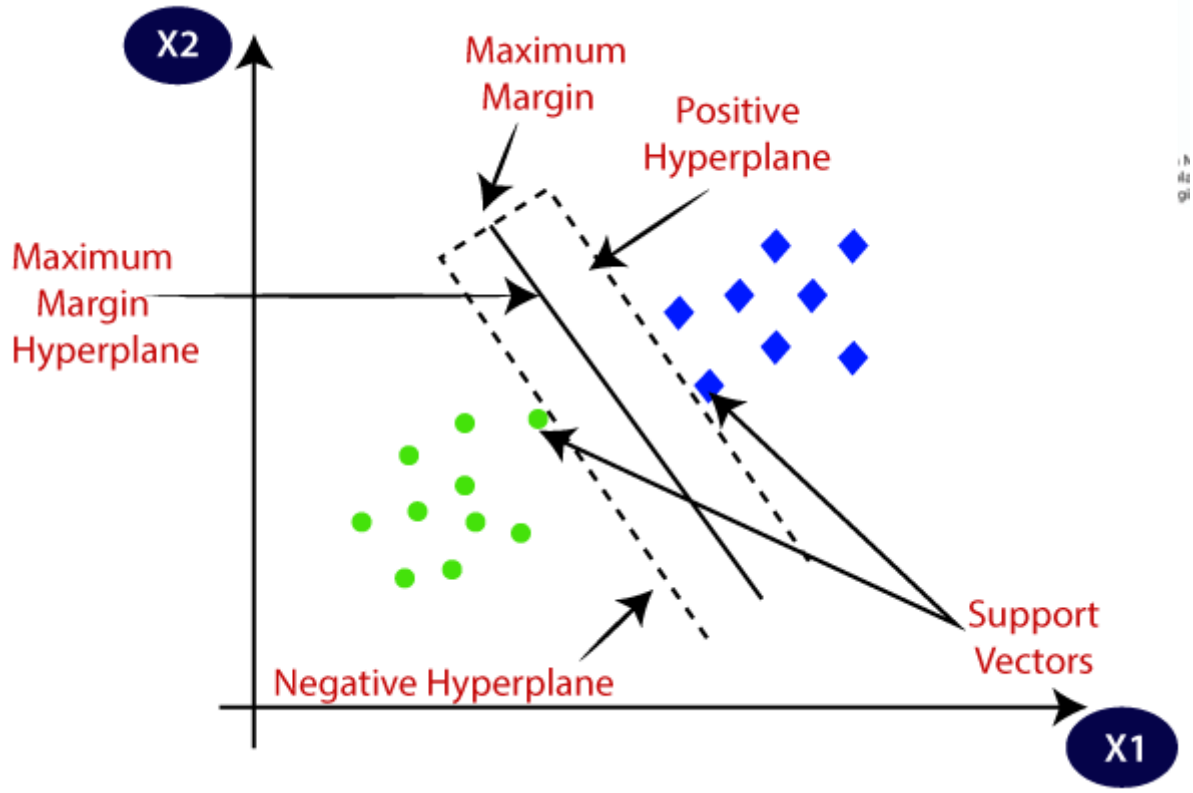
0.9998527245949926

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 4827 |
| 2 | 1.00 | 1.00 | 1.00 | 1963 |
| accuracy | | | 1.00 | 6790 |
| macro avg | 1.00 | 1.00 | 1.00 | 6790 |

weighted avg 1.00 1.00 1.00 6790

## 4.5 SUPPORT VECTOR MACHINE

- Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.
- The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.
- SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. Consider the below diagram in which there are two different categories that are classified using a decision boundary or hyperplane



**Types of SVM:**

**SVM can be of two types:**

- **Linear SVM:** Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.
- **Non-linear SVM:** Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

**Linear SVM:**

- The working of the SVM algorithm can be understood by using an example. Suppose we have a dataset that has two tags (green and blue), and the dataset has two features x1 and x2. We want a classifier that can classify the pair(x1, x2) of coordinates in either green or blue.
- So as it is 2-d space so by just using a straight line, we can easily separate these two classes. But there can be multiple lines that can separate these classes.
- Hence, the SVM algorithm helps to find the best line or decision boundary; this best boundary or region is called as a **hyperplane**. SVM algorithm finds the closest point of the lines from both the classes. These points are called support vectors. The distance between the vectors and the hyperplane is called as **margin**. And the goal of SVM is to maximize this margin. The **hyperplane** with maximum margin is called the **optimal hyperplane**.

**Non-Linear SVM:**

- If data is linearly **arranged**, then we can separate it by using a straight line, but for non-linear data, we cannot draw a single straight line.
- So to separate these data points, we need to add one more dimension. For linear data, we have used two dimensions x and y, so for non-linear data, we will add a third dimension z. It can be calculated as:

  $\mathbf{z=x^2 +y^2}$
- So now, SVM will divide the datasets into classes

- Since we are in 3-d Space, hence it is looking like a plane parallel to the x-axis. If we convert it in 2d space with z=1
- Hence we get a circumference of radius 1 in case of non-linear data.

### 4.5.1 Python Implementation Of Support Vector Machine:

we will pre-process/prepare the data so that we can use it in our code efficiently. It will be the same as we have done in Data pre-processing topic. The code for this is given below:

```
1. #Data Pre-procesing Step
2. # importing libraries
3. import numpy as nm
4. import matplotlib.pyplot as mp
5. import pandas as pd
6. #importing datasets
7. a=pd.read_excel(r"C:\Users\vennela\Documents\miniproject2/traindata1.
   xlsx",header=0)
```

By executing the above lines of code, we will get the dataset as the output. Consider the given image:

```
  Age of the patient Gender of the patient  Total Bilirubin  \
0                  65.0                Female              0.7
1                  62.0                  Male             10.9
2                  62.0                  Male              7.3
3                  58.0                  Male              1.0


Direct Bilirubin   Alkphos Alkaline Phosphotase  \
0                  0.1                          187.0
1                  5.5                          699.0
2                  4.1                          490.0
3                  0.4                          182.0

Sgpt Alamine Aminotransferase  Sgot Aspartate Aminotransferase  \
0                                 16.0                              18.0
1                                 64.0                             100.0
2                                 60.0                              68.0
3                                 14.0                              20.0

Total Protiens   ALB Albumin  A/G Ratio Albumin and Globulin Ratio \
0                 6.8          3.3                                 0.90
1                 7.5          3.2                                 0.74
2                 7.0          3.3                                 0.89
3                 6.8          3.4                                 1.00

Result
```

```
0           1
1           1
2           1
3           1
```

The categorical values present in the gender column is replaced with female as 0 and male as 1. Below is the code for it:

**a["Gender of the patient"]=a["Gender of the patient"].replace("Female",0)**

**a["Gender of the patient"]=a["Gender of the patient"].replace("Male",1)**

Checking and removing the null values present in the dataset. Below is the code for it:

**print(a.isnull().sum())**

```
Age of the patient                    2
Gender of the patient                 902
Total Bilirubin                       648
Direct Bilirubin                      561
 Alkphos Alkaline Phosphotase         796
 Sgpt Alamine Aminotransferase        538
Sgot Aspartate Aminotransferase       462
Total Protiens                        463
 ALB Albumin                          494
A/G Ratio Albumin and Globulin Ratio  559
Result                                0
dtype: int64
```

**a=a.dropna()**

**print(a.isnull().sum())**

```
Age of the patient                    0
Gender of the patient                 0
Total Bilirubin                       0
Direct Bilirubin                      0
 Alkphos Alkaline Phosphotase         0
 Sgpt Alamine Aminotransferase        0
Sgot Aspartate Aminotransferase       0
Total Protiens                        0
 ALB Albumin                          0
A/G Ratio Albumin and Globulin Ratio  0
Result                                0
dtype: int64
```

Now, we will extract the dependent and independent variables from the given dataset. Below is the code for it:

**#Extracting Independent and dependent Variable**

**x=a.iloc[:,2:-1]**

**y=np.array(a.iloc[:,10:])**

In the above code, we have taken [2, 3, 4, 5, 6, 7, 8, 9] for x because our independent Variables are **Total Bilirubin, Direct Bilirubin, Alkphos Alkaline Phosphotase , Sgpt Ala mine Aminotransferase, Sgot Aspartate Aminotransferase, Total Protiens, ALB Albumi n A/G Ratio Albumin and Globulin Ratio** , which are at index 2, 3, 4, 5, 6, 7, 8, 9. And we have taken 10 for y variable because our dependent variable is at index 10.

Now we will split the dataset into a training set and test set. Below is the code for it:

```
# Splitting the dataset into training and test set.
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test= train_test_split(x, y, random_state=0)
```

**Fitting the SVM classifier to the training set:**

Now the training set will be fitted to the SVM classifier. To create the SVM classifier, we wil l import **SVC** class from **Sklearn.svm** library. Below is the code for it:

```
from sklearn.svm import SVC
clf_s = SVC(random_state=7,gamma="auto")
clf_s.fit(X_train,Y_train)
```

Output follows as:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr',degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=0,
    shrinking=True, tol=0.001, verbose=False)
```

The model performance can be altered by changing the value of **C(Regularization factor), gamma, and kernel**.

**Predicting the test set result:**

Now, we will predict the output for test set. For this, we will create a new vector y_pred_s. Below is the code for it:

```
#Predicting the test set result
y_pred_s=clf_s.predict(X_test)
```

In the above code, we have created a y_pred_s vector to predict the test set result.

By executing the above code, a new vector (y_pred_s) will be created under the variable explorer option. It can be seen as:

**[1 1 1 ... 1 1 1]**

**Test accuracy of the result:**

If we want to know the number of correct and incorrect predictions, we need to use the confusion matrix. Below is the code for it:

4. **#Creating the Confusion matrix**
5. **from sklearn.metrics import confusion_matrix**
6. **print(confusion_matrix(y_pred_s, Y_test))**

output:

[[4824 2]
[ 1 1963]]

We will also create accuracy score and classification report to analyse classification. To create it, we need to import the **accuracy_score** function and **classification_report** function from the sklearn library. After importing the function, we will call it using a new variable **cm**. The function takes two parameters, mainly **Y_test** (the actual values) and y_**predict** (the targeted value return by the classifier). Below is the code for it:

- Accuracy is the percent of correct classifications and can be defined as:

$$\text{Accuracy} = (TP + TN/(TP + FP + TN + FN)) \times 100$$

- Sensitivity is the rate of true positive and can be defined as:

$$\text{Precision} = (TP/(TP + FN)) \times 100$$

- Specificity is the true negative rate and can be defined as:

$$\text{Recall} = (TN/(TN + FP)) \times 100$$

Where:

TP = the number of positive examples correctly classified.

FP = the number of positive examples misclassified as negative

FN = the number of negative examples misclassified as positive

TN = the number of negative examples correctly classified

```
5. #Creating the  Accuracy Score and classification report
6. from sklearn.metrics import accuracy_score ,classification_report
7. print(accuracy_score(y_pred_s,Y_test))
8. print(classification_report(y_pred_s,Y_test))
```

0.9998527245949926

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 1 | 1.00 | 1.00 | 1.00 | 4827 |
| 2 | 1.00 | 1.00 | 1.00 | 1963 |
| accuracy | | | 1.00 | 6790 |
| macro avg | 1.00 | 1.00 | 1.00 | 6790 |
| weighted avg | 1.00 | 1.00 | 1.00 | 6790 |

# 5 CONCLUSION

The main focus of this paper is to throw light on the importance of different classification techniques for disease prediction, particularly liver related diseases. The dataset that has been reviewed is considered in so many existing techniques.

In this project, we have proposed methods for predicting liver disease in patients using machine learning techniques. The four machine learning techniques that were used include SVM, Logistic Regression, Decision Tree and Random Forest. The system was implemented using all the models and their performance was evaluated. Performance evaluation was based on certain performance metrics. Decision Tree, Random Forest and Support Vector Machine was the model that resulted in the highest accuracy with an accuracy of 99%. Comparing this techniques with logistic regression with the high accuracy techniques, it was discovered that logistic regression proved poor efficient. This project can be improved by creating GUI and make available for all people.

# REFERENCES

**References:**

**Websites:**


1. **https://www.geeksforgeeks.org**
2. **https://www.youtube.com**
3. **https://www.javatpoint.com/machine-learning**
4. **https://www.kaggle.com/**


**Books:**


1. **Hands on machine learning with scikit learn and tenserflow**
2. **Introduction to machine learning with python**