VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590 018

A Mini Project Report on

"HEART DISEASE PREDICTION SYSTEM"

Submitted in partial fulfillment of the requirements as a part of the

AI/ML INTERNSHIP

(NASTECH)

For the award of degree of

Bachelor of Engineering
in
Information Science and Engineering

Submitted by

SUSHMITHA G — **1RN18IS111**

TANVI PRASAD — **1RN18IS113**

Internship Project Coordinators

Dr. R Rajkumar
Associate Professor
Dept. of ISE, RNSIT

Mr. Deepak Garg
Founder
NASTECH

Department of Information Science and Engineering

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,
Bengaluru – 560 098

2020 -2021

RNS Institute of Technology

Channasandra, Dr. Vishnuvardhan Road, RR Nagar Post,

Bengaluru – 560 098

DEPARTMENT OF INFORMATION SCIENCE & ENGINEERING

CERTIFICATE

This is to certify that the mini project report entitled HEART DISEASE PREDICTION SYSTEM has been successfully completed by Sushmitha G bearing USN 1RN18IS111 and Tanvi Prasad bearing USN 1RN18IS113 , presently VII semester students of RNS Institute of Technology in partial fulfillment of the requirements as a part of the ***AI/ML Internship (NASTECH)*** for the award of the degree of ***Bachelor of Engineering in Information Science and Engineering*** under **Visvesvaraya Technological University, Belagavi** during academic year **2021 – 2022**. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report and deposited in the departmental library. The mini project report has been approved as it satisfies the academic requirements as a part of Mobile.

Dr. R Rajkumar	**Mr. Deepak Garg**	**Dr. Suresh L**
Coordinator	Coordinator	Professor and HoD
Associate Professor		

External Viva

Name of the Examiners	**Signature with date**
1. _____	_____
2. _____	_____

ABSTRACT

The term heart disease, also called cardiovascular disease, encompasses the diverse diseases that affect the heart. The World Health Organization estimates that 12 million deaths occur worldwide every year due to heart disease. It is the major cause of deaths in many developing countries. For example, in the United States, it kills one person every 34 seconds. It is also the main cause of deaths in India, which proves that heart disease is one of the most dangerous diseases threatening adults lives today. Heart disease diagnosis is one of the most critical and challenging tasks in the healthcare field. It must be diagnosed quickly, efficiently and correctly in order to save lives. It requires the patient to do many tests, and healthcare professionals must carefully examine the results. That is researchers have been interested in predicting heart disease, and they developed different heart disease prediction systems using various machine learning algorithms. Some of them achieved better results than others. Many used the famous UCI heart disease dataset to train and test their classifier, while others used data obtained from other hospitals accessible to them.

This project provides an overview of the machine learning classification techniques used in the field of diagnosing heart disease, and how previous researchers implemented them. It throws the light on how important is machine learning in the healthcare field and how it can make accurate predictions and help healthcare professionals.

ACKNOWLEDGMENT

At the very onset I would like to place our gratefulness to all those people who helped me in making the Internship a successful one.

Coming up, this internship to be a success was not easy. Apart from the sheer effort, the enlightenment of the very experienced teachers also plays a paramount role because it is they who guided me in the right direction.

First of all, I would like to thank the Management of **RNS Institute of Technology** for providing such a healthy environment for the successful completion of internship work.

In this regard, I express sincere gratitude to our beloved **Principal Dr. M K Venkatesha**, for providing us all the facilities.

We are extremely grateful to our own and beloved Professor and Head of Department of Information science and Engineering, **Dr. Suresh L**, for having accepted to patronize me in the right direction with all her wisdom.

We place our heartfelt thanks to **Dr. S Sathish Kumar**, Professor, Department of Information Science and Engineering, for having guided internship and the staff members of the department of Information Science and Engineering for helping at all times.

I also thank our internship coordinator **Dr. R Rajkumar**, Associate Professor, Department of Information Science and Engineering. I would thank my friends for having supported me with all their strength and might. Last but not the least, I thank my parents for supporting and encouraging me throughout. I have made an honest effort in this assignment.

SUSHMITHA G TANVI PRASAD

1RN18IS111 1RN18IS113

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF GRAPHS

INTRODUCTION

1.1Organization

1.1.1 Company Profile

NASTECH is formed with the purpose of bridging the gap between Academia and Industry Nastech is one of the leading Global Certification and Training service providers for technical and management programs for educational institutions. We collaborate with educational institutes to understand their requirements and form a strategy in consultation with all stakeholders to fulfill those by skilling, reskilling and upskilling the students and faculties on new age skills and technologies.

1.1.2 Domain/Technology

Artificial intelligence (AI) is a part of computer science that has the task of making computers more intelligent. Since the most basic requirement of intelligence is learning, hence came the subfield of AI that is called machine learning (ML). ML is one of the most rapidly evolving fields of AI which is used in many areas of life, primarily in the healthcare field. ML has a great value in the healthcare field since it is an intelligent tool to analyze data, and the medical field is rich with data. This project uses different types of ML techniques.

1.1.3 Department

R.N.Shetty Institute of Technology (RNSIT) established in the year 2001, is the brain-child of the Group Chairman, Dr. R. N. Shetty. The Murudeshwar Group of Companies headed by Sri. R. N. Shetty is a leading player in many industries viz construction, manufacturing, hotel, automobile, power & IT services and education. The group has contributed significantly to the field of education. A number of educational institutions are run by the R. N. Shetty Trust, RNSIT being one amongst them. With a continuous desire to provide quality education to the society, the group has established RNSIT, an institution to nourish and produce the best of engineering talents in the country. RNSIT is one of the best and top accredited engineering colleges in Bengaluru.

1.2 Problem Statement

Following is the discuss regarding heart disease prediction based on Existing system and their limitations and proposed model.

1.2.1 Existing System and Their Limitations

Lot of work has been carried out to predict heart disease using UCI Machine Learning dataset. Different levels of accuracy have been attained using various data mining techniques which are explained as follows. Avinash Golande and et. al.;studies various different ML algorithms that can be used for classification of heart disease. Research was carried out to study Decision Tree, KNN and K-Means algorithms that can be used for classification and their accuracy were compared [1]. This research concludes that accuracy obtained by Decision Tree was highest further it was inferred that it can be made efficient by combination of different techniques and parameter tuning.

T.Nagamani, et al. have proposed a system [2] which deployed data mining techniques along with the MapReduce algorithm. The accuracy obtained according to this paper for the 45 instances of testing set, was greater than the accuracy obtained using conventional fuzzy artificial neural network. Here, the accuracy of algorithm used was improved due to use of dynamic schema and linear scaling.

Fahd Saleh Alotaibi has designed a ML model comparing five different algorithms [3]. Rapid Miner tool was used which resulted in higher accuracy compared to Matlab and Weka tool. In this research the accuracy of Decision Tree, Logistic Regression, Random Forest, Naive Bayes and SVM classification algorithms were compared. Decision tree algorithm had the highest accuracy.

Anjan Nikhil Repaka, ea tl., proposed a system in [4] that uses NB (Naïve Bayesian) techniques for classification of dataset and AES (Advanced Encryption Standard) algorithm for secure data transfer for prediction of disease. Theresa Princy. R, et al, executed a survey including different classification algorithm used for predicting heart disease. The classification techniques used were Naive Bayes, KNN (KNearest Neighbour), Decision tree, Neural network and accuracy of the classifiers was analyzed for different number of attributes [5].

Nagaraj M Lutimath, et al., has performed the heart disease prediction using Naive bayes classification and SVM (Support Vector Machine). The performance measures used in analysis are Mean Absolute Error, Sum of Squared Error and Root Mean Squared Error, it is established that SVM was emerged as superior algorithm in terms of accuracy over Naive Bayes [6].

The main idea behind the proposed system after reviewing the above papers was to create a heart disease prediction system based on the inputs as shown in Table 1. We analyzed the classification algorithms namely Decision Tree, Random Forest, Logistic Regression and Naive Bayes based on their Accuracy, Precision, Recall and f-measure scores and identified the best classification algorithm which can be used in the heart disease prediction

1.2.2 Proposed Model

The proposed work predicts heart disease by exploring the above mentioned eight classification algorithms and does performance analysis. The objective of this study is to effectively predict if the patient suffers from heart disease. The health professional enters the input values from the patient's health report. The data is fed into a model which predicts the probability of having heart disease.

A. Data Collection and Preprocessing

The dataset used is the Heart disease Dataset which is a combination of 4 different databases, but only the UCI Cleveland dataset was used. This database consists of a total of 76 attributes but all published experiments refer to using a subset of only 14 features

B. Classification

The attributes mentioned in Table 1 are provided as input to the different ML algorithms such as Random Forest, Decision Tree, Logistic Regression and Naive Bayes classification techniques [12]. The input dataset is split into 80% of the training dataset and the remaining 20% into the test dataset. Training dataset is the dataset which is used to train a model. Testing dataset is used to check the performance of the trained model. For each of the algorithms the performance is computed and analyzed based on different metrics used such as accuracy, precision, recall and F-measure scores as described further. The different algorithms explored in this paper are listed as below.

a) Random Forest Random Forest algorithms are used for classification as well as regression. It creates a tree for the data and makes prediction based on that. Random Forest algorithm can be used on large datasets and can produce the same result even when large sets record values are missing. The generated samples from the decision tree can be saved so that it can be used on other data. In random forest there are two stages, firstly create a random forest then make a prediction using a random forest classifier created in the first stage.

b) Decision Tree Decision Tree algorithm is in the form of a flowchart where the inner node represents the dataset attributes and the outer branches are the outcome. Decision Tree is chosen because they are fast, reliable, easy to interpret and very little data preparation is required. In Decision Tree, the prediction of class label originates from root of the tree. The value of the root attribute is compared to record's attribute. On the result of comparison, the corresponding branch is followed to that value and jump is made to the next node.

c) Logistic Regression Logistic Regression is a classification algorithm mostly used for binary classification problems. In logistic regression instead of fitting a straight line or hyper plane, the logistic regression algorithm uses the logistic function to squeeze the output of a linear equation between 0 and 1. There are 13 independent variables which makes logistic regression good for classification.

d) Naive Bayes Naïve Bayes algorithm is based on the Bayes rule. The independence between the attributes of the dataset is the main assumption and the most important in making a classification. It is easy and fast to predict and holds best when the assumption of independence holds. Bayes' theorem calculates the posterior probability of an event (A) given some prior probability of event B represented by P(A/B)[10] as shown in equation 1: $P(A|B) = (P(B|A)\ P(A)) / P(B)$

e) "Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

f) K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification of predictive problems in industry. The following two properties would define KNN well: Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification. Non-parametric learning algorithm − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

g) XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured/tabular data, decision tree-based algorithms are considered best-in-class right now.

h) A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.

1.2.3 Program Formulation

The proposed work predicts heart disease by exploring the above mentioned eight classification algorithms and does performance analysis. The objective of this study is to effectively predict if the patient suffers from heart disease. The health professional enters the input values from the patient's health report. The data is fed into a model which predicts the probability of having heart disease.

REQUIREMENT ANALYSIS TOOLS AND TECHNOLOGY

We have discussed tools and technologies discussed in this project.

2.1 Hardware and Software Requirements

CoLab system requirements

Memory: 8 GB

Graphics Card: AMD Radeon RX 480

CPU: Intel Core i5-4590

File Size: 2 GB

OS: Windows 7 SP1

CoLab minimum requirements

Memory: 4 GB

Graphics Card: NVIDIA GeForce GTX 970

CPU: Intel Core i5-4590

File Size: 2 GB

OS: Windows 7 SP1

2.2 Tools/Language/Platform

The project, prediction of heart disease was built using google colab. The dataset is based on UCI heart Disease Data Set and we have 303 instances. According to UCI, “This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them.

Language used in this project is python.

DESIGN AND IMPLEMENTATION

This chapter focuses on the design of the project.

3.1 ARCHITECTURE AND FLOW CHART

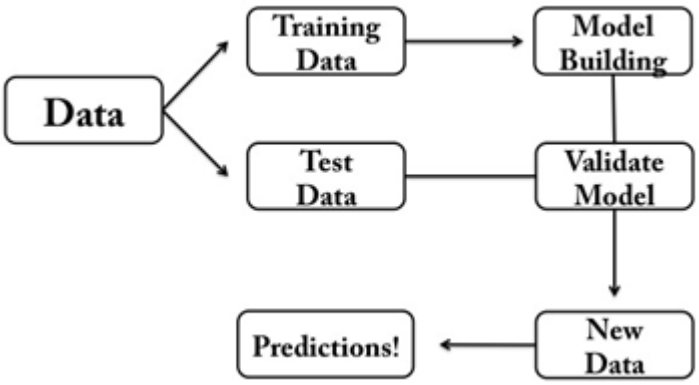

Fig 3.1.1 Simple Architecture of Machine Learning

Heart disease data is pre-processed after collection of various records. The dataset contains a total of 303 patient records, where 6 records are with some missing values. Those 6 records have been removed from the dataset and the remaining 297 patient records are used in pre-processing. The multiclass variable and binary classification are introduced for the attributes of the given dataset. The multi-class variable is used to check the presence or absence of heart disease. In the instance of the patient having heart disease, the value is set 1, else the value is set to 0 indicating the absence of heart disease in the patient. The pre-processing of data is carried out by converting medical records into diagnosis values. The results of data pre-processing for 297 patient records indicate that 137 records show the value of 1 establishing the presence of heart disease while the remaining 160 reflected the value of 0 indicating the absence of heart disease.

From among the 13 attributes of the data set, two attributes pertaining to age and sex are used to identify the personal information of the patient. The remaining 11 attributes are considered important as they contain vital clinical records. Clinical records are vital to diagnosis and learning the severity of heart disease. As previously mentioned in this experiment, several (ML) techniques are used

namely, NB, GLM, LR, DL, DT, RF, GBT and SVM. The experiment was repeated with all the ML techniques using all 13 attributes

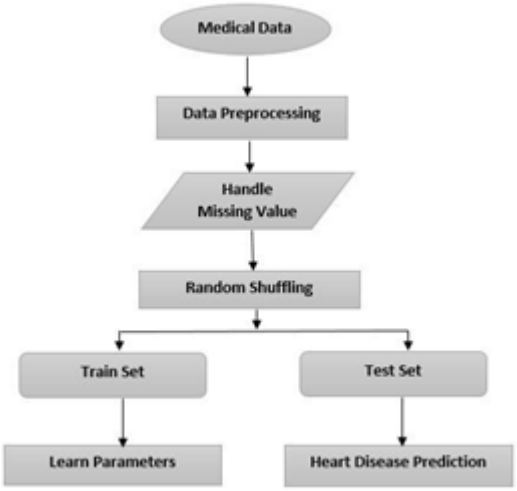

Fig 3.1.2 Heart Disease Prediction Model

3.2 PROBLEM STATEMENNT

The proposed work predicts heart disease by exploring the above mentioned eight classification algorithms and does performance analysis. The objective of this study is to effectively predict if the patient suffers from heart disease. The health professional enters the input values from the patient's health report. The data is fed into a model which predicts the probability of having heart disease.

DATASET

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

3.3 Algorithm / Pseudocode

Following are the algorithms' pseudocode used in the project.

Logistic Regression

Input: Training data

1. For i←1 to k
2. For each training data instance d_i:
3. Set the target value for the regression to
   $z_i \leftarrow \frac{y_j - P(1 \mid d_j)}{[P(1 \mid d_j) . (1 - P(1 \mid d_j))]}$
4. nitialize the weight of instance d_j to $P(1 \mid d_j). (1 - P(1 \mid d_j)$
5. finalize a f(j) to the data with class value (z_j) & weights (wj)
   Classification Label Decision
6. Assign (class label:1) if $P(1|d_j) > 0.5$, otherwise (class label: 2)

Fig 3.3.1 Pseudocode of Logistic Regression

Logistic regression is basically a supervised classification algorithm. In a classification problem, the target variable (or output), y, can take only discrete values for a given set of features (or inputs), X. Contrary to popular belief, logistic regression IS a regression model. The model builds a regression model to predict the probability that a given data entry belongs to the category numbered as “1”. Logistic regression becomes a classification technique only when a decision threshold is brought into the picture. The setting of the threshold value is a very important aspect of Logistic regression and is dependent on the classification problem itself.

Naive Bayes

Input:

Training dataset T,

F= (f_1, f_2, f_3,.., f_n) // value of the predictor variable in testing dataset.

Output:

A class of testing dataset.

Step:

1. Read the training dataset T;
2. Calculate the mean and standard deviation of the predictor variables in each class;
3. Repeat

   Calculate the probability of f_i using the gauss density equation in each class;

   Until the probability of all predictor variables (f_1, f_2, f_3,.., f_n) has been calculated.
4. Calculate the likelihood for each class;
5. Get the greatest likelihood;

Fig 3.3.2 Pseudocode of Naive Bayes

Naive Bayes is among one of the very simple and powerful algorithms for classification based on **Bayes Theorem** with an assumption of independence among the predictors. The Naive Bayes classifier assumes that the presence of a feature in a class is not related to any other feature. Naive Bayes is a classification algorithm for binary and multi-class classification problems. Based on prior knowledge of conditions that may be related to an event, Bayes theorem describes the probability of the event. Conditional probability can be found this way.

SVM

Support Vector Machine (SVM) is a relatively simple **Supervised Machine Learning Algorithm** used for classification and/or regression. It is more preferred for classification but is sometimes very useful for regression as well. Basically, SVM finds a hyper-plane that creates a boundary between the

types of data. In 2-dimensional space, this hyper-plane is nothing but a line. In SVM, we plot each data item in the dataset in an N-dimensional space, where N is the number of features/attributes in the data. Next, find the optimal hyperplane to separate the data. So, by this, you must have understood that inherently, SVM can only perform binary classification (i.e., choose between two classes). However, there are various techniques to use for multi-class problems.

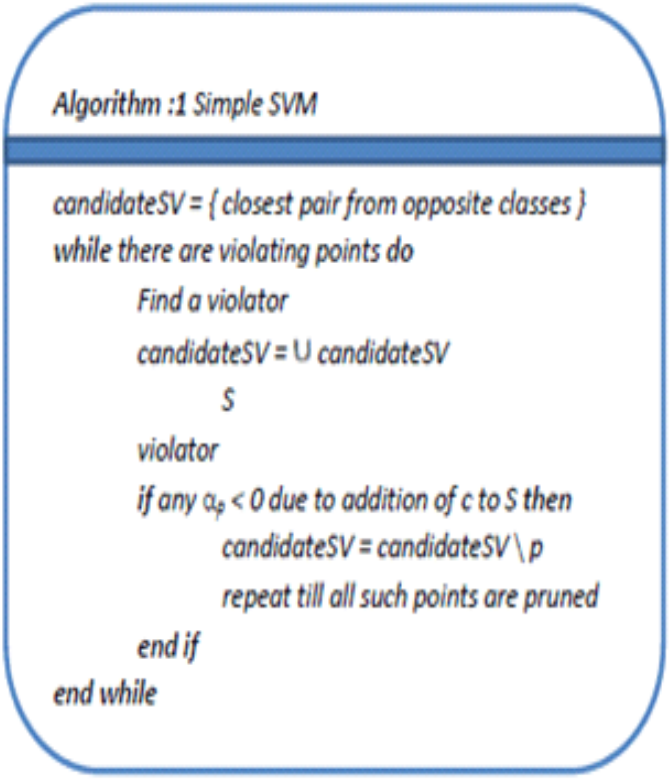

Fig 3.3.3 Pseudo Code for SVM

K Nearest Neighbour

K-Nearest Neighbours is one of the most basic yet essential classification algorithms in Machine Learning. It belongs to the supervised learning domain and finds intense application in pattern recognition, data mining and intrusion detection. It is widely disposable in real-life scenarios since it is non-parametric, meaning, it does not make any underlying assumptions about the distribution of data (as opposed to other algorithms such as GMM, which assume a Gaussian distribution of the

given data). We are given some prior data (also called training data), which classifies coordinates into groups identified by an attribute.

Algorithm 4: *kNN kernel* Algorithm

Input : D, a chunk of the original distance matrix;
$n_{chunksize}$, dimension of the chunk;
split, index of split;
chunk, index of chunk;
Maxk, an array to hold the farthest neighbors for each row index in the chunks;

Output: none, an (intermediate) *k*NN graph stored in Gk′;

```
1  row′ ← blockIdx.x × blockDim.x + threadIdx.x;
2  if row′ < n_chunksize then
3      row ← split × n_chunkSize + row′ // absolute row in the distance matrix;
4      for column′ ← 1 to n_chunksize do
5          column ← chunk × n_chunkSize + column′ //absolute column in the distance matrix;
6          if row = column or row > n_row or column > n_col then
7              continue /* exclude diagonal and pad regions */;
8          if D[row′ × n_chunksize + column′] < Gk′[Maxk[row′]].weight then
9              Gk′[Maxk[row′]].source ← row;
10             Gk′[Maxk[row′]].target ← column;
11             Gk′[Maxk[row′]].weight ← D[row′ × n_chunksize + column′];
12             Search the new maximum element in row′(D) and store the index in Maxk[row′];
```

Fig 3.3.4 K Nearest Neighbours

Decision Tree

Decision Trees are a type of Supervised Machine Learning (that is you explain what the input is and what the corresponding output is in the training data) where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves. The leaves are the decisions or the final outcomes. And the decision nodes are where the data is split.

GenDecTree(Sample S, Features F)

Steps:

1. ***If*** *stopping_condition(S, F) = true* ***then***
   a. *Leaf = createNode()*
   b. *leafLabel = classify(s)*
   c. ***return*** *leaf*
2. *root = createNode()*
3. *root.test_condition =findBestSpilt(S,F)*
4. *V = {v | v a possible outcomecfroot.test_condition}*
5. ***For each*** *value v ∈ V:*
   a. S_v *= {s | root.test_condition(s) = v and s ∈ S };*
   b. *Child = TreeGrowth (*S_v*, F);*
   c. *Add child as descent of root and label the edge {root → child} as v*
6. ***return*** *root*

Fig 3.3.5 Pseudocode Decision Tree

Random Forest

A random forest is a machine learning technique that's used to solve regression and classification problems. It utilizes ensemble learning, which is a technique that combines many classifiers to provide solutions to complex problems. A random forest algorithm consists of many decision trees. The 'forest' generated by the random forest algorithm is trained through bagging or bootstrap aggregating. Bagging is an ensemble meta-algorithm that improves the accuracy of machine learning algorithms. The (random forest) algorithm establishes the outcome based on the predictions of the decision trees. It predicts by taking the average or mean of the output from various trees. Increasing the number of trees increases the precision of the outcome.

Algorithm 1: Pseudo code for the random forest algorithm

To generate c classifiers:
for i = 1 to c **do**
  Randomly sample the training data D with replacement to produce D_i
  Create a root node, N_i containing D_i
  Call BuildTree(N_i)
end for

BuildTree(N):
if N contains instances of only one class **then**
  return
else
  Randomly select x% of the possible splitting features in N
  Select the feature F with the highest information gain to split on
  Create f child nodes of N, N_1,..., N_f, where F has f possible values (F_1, ... , F_f)
  for i = 1 to f **do**
    Set the contents of N_i to D_i, where D_i is all instances in N that match
    F_i
    Call BuildTree(N_i)
  end for
end if

Fig 3.3.6 pseudocode for Random Forest

XG Boost

XGBoost is an algorithm that has recently been dominating applied machine learning and Kaggle competitions for structured or tabular data. XGBoost is an implementation of gradient boosted decision trees designed for speed and performance.

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For $m = 1$ to M:

1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.

3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

Fig 3.3.6 Pseudocode of XG Boost Algorithm

NEURAL NETWORK

CNN's were first developed and used around the 1980s. The most that a CNN could do at that time was recognize handwritten digits. It was mostly used in the postal sectors to read zip codes, pin codes, etc. The important thing to remember about any deep learning model is that it requires a large amount of data to train and also requires a lot of computing resources. This was a major drawback for CNNs at that period and hence CNNs were only limited to the postal sectors and it failed to enter the world of machine learning.

Algorithm 1 Pseudo code for a convolutional layer

```
for i from 1 to m do                    —inter-output
    for j from 1 to n do                —intra-output
        for r from 1 to R_o do
            for c from 1 to R_o do
                tmp = 0
                for ii from 1 to k do
                    for jj from 1 to k do
                        tmp = tmp + K[ii][jj] × X[j][s × (r − 1) + ii][s × (c − 1) + j
                    end for
                end for
                Y[i][r][c] = Y[i][r][c] + tmp
                if j == n
                    Y[i][r][c] = f(Y[i][r][c] + bias)
                end if
            end for
        end for
    end for
end for
```

Fig 3.3.7 Pseudocode of CNN

3.4 LIBRARY USED

Following are the functional modules used in this project: -

(a) NumPy - **NumPy** is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays The ancestor of NumPy, Numeric, was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. NumPy is open-source software and has many contributors. NumPy is a NumFOCUS fiscally sponsored project.

(b) Pandas - **pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating

numerical tables and time series. It is free software released under the three-clause BSD license. The name is derived from the term "panel data", an econometrics term for data sets that include observations over multiple time periods for the same individuals. Its name is a play on the phrase "Python data analysis" itself. Wes McKinney started building what would become pandas at AQR Capital while he was a researcher there from 2007 to 2010.

(c) Matplotlib - Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK. There is also a procedural "pylab" interface based on a state machine, designed to closely resemble that of MATLAB, though its use is discouraged. SciPy makes use of Matplotlib. Matplotlib was originally written by John D. Hunter.

(d) Seaborn - Seaborn is an amazing visualization library for statistical graphics plotting in Python. It provides beautiful default styles and color palettes to make statistical plots more attractive. It is built on the top of matplotlib library and also closely integrated to the data structures from pandas. Seaborn aims to make visualization the central part of exploring and understanding data. It provides dataset-oriented APIs, so that we can switch between different visual representations for the same variables for better understanding of the dataset.

(e) os - The **OS** module in Python provides functions for interacting with the operating system. OS comes under Python’s standard utility modules. This module provides a portable way of using operating system-dependent functionality. The *os* and *os.path* modules include many functions to interact with the file system.

(f) warnings - **Warning** messages are typically issued in situations where it is useful to alert the user of some condition in a program, where that condition (normally) doesn’t warrant raising an exception and terminating the program. For example, one might want to issue a warning when a program uses an obsolete module. Warning messages are normally written to sys.stderr, but their disposition can be changed flexibly, from ignoring all warnings to turning them into exceptions. The disposition of warnings can vary based on the warning category, the text of the warning message, and the source location where it is issued. Repetitions of a particular warning for the same source location are typically suppressed.

DATASET

This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date. The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

3.5 CONFIGURATION

The dataset used is a multivariate type of dataset which means providing or involving a variety of separate mathematical or statistical variables, multivariate numerical data analysis. It is composed of 14 attributes which are age, sex, chest pain type, resting blood pressure, serum cholesterol, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercise induced angina, old peak — ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels and Thalassemia. This database includes 76 attributes, but all published studies relate to the use of a subset of 14 of them. The Cleveland database is the only one used by ML researchers to date. One of the major tasks on this dataset is to predict based on the given attributes of a patient that whether that particular person has a heart disease or not and other is the experimental task to diagnose and find out various insights from this dataset which could help in understanding the problem more. We have first imported all the modules in our project. Then we have described the dataset.

Then, we have done the exploratory data analytics of the dataset. Exploratory data analysis (EDA) is used by data scientists to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods. It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.

Description

In [7]:
```
dataset.describe()
```

Out[7]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.366337	0.683168	0.966997	131.623762	246.264026	0.148515	0.528053	149.646865	0.326733	1.039604	1.399340	0.729373	2.313531	0.544554
std	9.082101	0.466011	1.032052	17.538143	51.830751	0.356198	0.525860	22.905161	0.469794	1.161075	0.616226	1.022606	0.612277	0.498835
min	29.000000	0.000000	0.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	47.500000	0.000000	0.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000	0.000000	2.000000	0.000000
50%	55.000000	1.000000	1.000000	130.000000	240.000000	0.000000	1.000000	153.000000	0.000000	0.800000	1.000000	0.000000	2.000000	1.000000
75%	61.000000	1.000000	2.000000	140.000000	274.500000	0.000000	1.000000	166.000000	1.000000	1.600000	2.000000	1.000000	3.000000	1.000000
max	77.000000	1.000000	3.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	2.000000	4.000000	3.000000	1.000000

Fig 3.5.1 heart disease dataset description

In [8]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
age         303 non-null int64
sex         303 non-null int64
cp          303 non-null int64
trestbps    303 non-null int64
chol        303 non-null int64
fbs         303 non-null int64
restecg     303 non-null int64
thalach     303 non-null int64
exang       303 non-null int64
oldpeak     303 non-null float64
slope       303 non-null int64
ca          303 non-null int64
thal        303 non-null int64
target      303 non-null int64
dtypes: float64(1), int64(13)
memory usage: 33.2 KB
```

Fig 3.5.2 information regarding dataset

EDA is primarily used to see what data can reveal beyond the formal modeling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them. It can also help determine if the statistical techniques you are considering for data analysis are appropriate. Originally developed by American mathematician John Tukey in the 1970s, EDA techniques continue to be a widely used method in the data discovery process today.

The main purpose of EDA is to help look at data before making any assumptions. It can help identify obvious errors, as well as better understand patterns within the data, detect outliers or anomalous events, find interesting relations among the variables.

Data scientists can use exploratory analysis to ensure the results they produce are valid and applicable to any desired business outcomes and goals. EDA also helps stakeholders by confirming they are asking the right questions. EDA can help answer questions about standard deviations, categorical variables, and confidence intervals. Once EDA is complete and insights are drawn, its features can then be used for more sophisticated data analysis or modeling, including machine learning.

Then we have used train-test split approach. The train-test split procedure is used to estimate the performance of machine learning algorithms when they are used to make predictions on data not used to train the model.

It is a fast and easy procedure to perform, the results of which allow you to compare the performance of machine learning algorithms for your predictive modeling problem. Although simple to use and interpret, there are times when the procedure should not be used, such as when you have a small dataset and situations where additional configuration is required, such as when it is used for classification and the dataset is not balanced.

Splitting your dataset is essential for an unbiased evaluation of prediction performance. In most cases, it's enough to split your dataset randomly into three subsets:

- **The training set** is applied to train, or **fit**, your model. For example, you use the training set to find the optimal weights, or coefficients, for linear regression, logistic regression, or neural networks.
- **The validation set** is used for unbiased model evaluation during hyperparameter tuning. For example, when you want to find the optimal number of neurons in a neural network or the best kernel for a support vector machine, you experiment with different values. For each considered setting of hyperparameters, you fit the model with the training set and assess its performance with the validation set.
- **The test set** is needed for an unbiased evaluation of the final model.

We have used version 0.23.1 of **scikit-learn**, or **sklearn**. It has many packages for data science and machine learning, but for this project we focus on the **model_selection** package, specifically on the function **train_test_split()**.

It can be installed with pip install:

$ python -m pip install -U "scikit-learn==0.23.1"

OBSERVATIONS AND RESULTS

This chapter focuses on the design of the project.

4.1 TESTING

One of the key aspects of supervised machine learning is model evaluation and validation. When you evaluate the predictive performance of your model, it's essential that the process be unbiased. Using **train_test_split()** from the data science library scikit-learn, we can split dataset into subsets that minimize the potential for bias in your evaluation and validation process.

Supervised machine learning is about creating models that precisely map the given inputs (independent variables, or predictors) to the given outputs (dependent variables, or responses).

In regression analysis, typically the coefficient of determination, root-mean-square error, mean absolute error, or similar quantities are used. For classification problems, you often apply accuracy, precision, recall, F1, score, and related indicators.

Training, Validations and Test Sets

Splitting the dataset is essential for an unbiased evaluation of prediction performance. In most cases, it's enough to split your dataset randomly into three subsets:

a. **The training set** is applied to train, or **fit**, your model. For example, you use the training set to find the optimal weights, or coefficients, for linear regression, logistic regression, or neural networks.
b. **The validation set** is used for unbiased model evaluation during hyperparameter tuning. For example, when you want to find the optimal number of neurons in a neural network or the best kernel for a support vector machine, you experiment with different values. For each considered setting of hyperparameters, you fit the model with the training set and assess its performance with the validation set.
c. **The test set** is needed for an unbiased evaluation of the final model.

In less complex cases, when you don't have to tune hyperparameters, it's okay to work with only the training and test sets.

Underfitting and Overfitting

Splitting a dataset might also be important for detecting if your model suffers from one of two very common problems, called underfitting and overfitting:

i. **Underfitting** is usually the consequence of a model being unable to encapsulate the relations among data. For example, this can happen when trying to represent nonlinear relations with a linear model. Underfitted models will likely have poor performance with both training and test sets.
ii. **Overfitting** usually takes place when a model has an excessively complex structure and learns both the existing relations among data and noise. Such models often have bad generalization capabilities. Although they work well with training data, they usually yield poor performance with unseen (test) data.

Prerequisites for Using train_test_split()

We have used version 0.23.1 of **scikit-learn**, or **sklearn**. It has many packages for data science and machine learning, but for this tutorial you'll focus on the **model_selection** package, specifically on the function **train_test_split()**.

It can be installed with pip install:

$ python -m pip install -U "scikit-learn==0.23.1"

If in Anaconda, then it's probably already had it installed. However, if you want to use a fresh environment, ensure that you have the specified version, or use Miniconda, then you can install sklearn from Anaconda Cloud with conda install:

$ conda install -c anaconda scikit-learn = 0.23

4.2 Results

We have used eight algorithms to see which one is the most efficient one. First, we have logistic regression.

Logistic Regression - Logistic Regression is a classification algorithm mostly used for binary classification problems. In logistic regression instead of fitting a straight line or hyper plane, the

logistic regression algorithm uses the logistic function to squeeze the output of a linear equation between 0 and 1. There are 13 independent variables which makes logistic regression good for classification.

Logistic Regression

```
In [44]: from sklearn.linear_model import LogisticRegression

         lr = LogisticRegression()

         lr.fit(X_train,Y_train)

         Y_pred_lr = lr.predict(X_test)

In [45]: Y_pred_lr.shape

Out[45]: (61,)

In [46]: score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

         print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")

The accuracy score achieved using Logistic Regression is: 85.25 %
```

Fig 4.2.1 Logistic Regression accuracy

The efficiency of the logistic regression is 85.25 %.

Naïve Bayes - Naïve Bayes algorithm is based on the Bayes rule. The independence between the attributes of the dataset is the main assumption and the most important in making a classification. It is easy and fast to predict and holds best when the assumption of independence holds. Bayes' theorem calculates the posterior probability of an event (A) given some prior probability of event B represented by P(A/B) [10] as shown in equation 1: P(A|B) = (P(B|A)P(A)) / P(B).

Naive Bayes

```
[n [47]:
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train,Y_train)
Y_pred_nb = nb.predict(X_test)

[n [48]:
Y_pred_nb.shape

)ut[48]:
(61,)

[n [49]:
score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)
print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")

The accuracy score achieved using Naive Bayes is: 85.25 %
```

Fig 4.2.2 Naïve Bayes accuracy

The efficiency of naïve bayes is 85.25 %.

SVM - Support Vector Machine" (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.

SVM

```
:n [50]:
from sklearn import svm
sv = svm.SVC(kernel='linear')
sv.fit(X_train, Y_train)
Y_pred_svm = sv.predict(X_test)

:n [51]:
Y_pred_svm.shape

)ut[51]:
(61,)

:n [52]:
score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)
print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")

The accuracy score achieved using Linear SVM is: 81.97 %
```

Fig 4.2.3 SVM accuracy

K Nearest Neighbors - K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification of predictive problems in industry. The following two properties would define KNN well.

(a) Lazy learning algorithm − KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.

(b) Non-parametric learning algorithm − KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Fig 4.2.4 KNN accuracy

Decision Tree - Decision Tree algorithm is in the form of a flowchart where the inner node represents the dataset attributes and the outer branches are the outcome. Decision Tree is chosen because they are fast, reliable, easy to interpret and very little data preparation is required. In Decision Tree, the prediction of class label originates from root of the tree. The value of the root attribute is compared to record's attribute. On the result of comparison, the corresponding branch is followed to that value and jump is made to the next node.

Decision Tree

In [56]:
```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0


for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

In [57]:
```
print(Y_pred_dt.shape)
```

```
(61,)
```

In [58]:
```
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

Fig 4.2.5 Decision Tree accuracy

The efficiency of decision tree is 81.97 %.

Random Forest - Random Forest algorithms are used for classification as well as regression. It creates a tree for the data and makes prediction based on that. Random Forest algorithm can be used on large datasets and can produce the same result even when large sets record values are missing. The generated samples from the decision tree can be saved so that it can be used on other data. In random forest there are two stages, firstly create a random forest then make a prediction using a random forest classifier created in the first stage. Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie *et al.*, "because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate".

Random Forest

In [59]:
```
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0


for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

In [60]:
```
Y_pred_rf.shape
```

Out[60]: (61,)

In [61]:
```
score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_rf)+" %")
```

Fig 4.2.6 Random Forest accuracy

The efficiency of Random Forest is 95.08%.

XGBoost - XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework. In prediction problems involving unstructured data (images, text, etc.) artificial neural networks tend to outperform all other algorithms or frameworks. However, when it comes to small-to-medium structured / tabular data, decision tree-based algorithms are considered best in class right now. XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way. The same code runs on major distributed environment (Hadoop, SGE, MPI) and can solve problems beyond billions of examples.

XGBoost

In [62]:
```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)
```

In [63]:
```
Y_pred_xgb.shape
```

Out[63]:
```
(61,)
```

In [64]:
```
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 85.25 %
```

Fig 4.2.7 XGBoost accuracy

The efficiency of XGBoost is 85.25%.

Neural Network - A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input; so, the network generates the best possible result without needing to redesign the output criteria. The concept of neural networks, which has its roots in artificial intelligence, is swiftly gaining popularity in the development of trading systems.The architecture of the proposed convolutional neural network (CNN) is a feedforward network which works on a sequential single–input–single–output fashion. For binary classification experimentation, we assume that patients with the presence of CHD will be classified as ‘1’, and others (with CHD absent) will be classified as ‘0’. An experiment multi-class classification is also performed will be discussed later. As mentioned earlier, the number of active CHD attributes (phenotypes) obtained from the majority voting algorithm is 14. Let us proceed with an assumption that the number of training examples is N, so the input layer is $R^{N\times14}$ dimension.

Neural Network

In [65]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

In [66]:
```
# https://stats.stackexchange.com/a/136542 helped a lot in avoiding overfitting

model = Sequential()
model.add(Dense(11,activation='relu',input_dim=13))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

In [67]:
```
model.fit(X_train,Y_train,epochs=300)
```

```
Epoch 1/300
242/242 [==============================] - 3s 10ms/step - loss: 4.7817 - acc: 0.6074
Epoch 2/300
242/242 [==============================] - 0s 123us/step - loss: 2.6633 - acc: 0.6033
Epoch 3/300
242/242 [==============================] - 0s 124us/step - loss: 2.3371 - acc: 0.6281
Epoch 4/300
242/242 [==============================] - 0s 127us/step - loss: 2.2938 - acc: 0.6488
Epoch 5/300
242/242 [==============================] - 0s 127us/step - loss: 2.0712 - acc: 0.6860
Epoch 6/300
242/242 [==============================] - 0s 125us/step - loss: 2.0645 - acc: 0.6653
Epoch 7/300
242/242 [==============================] - 0s 124us/step - loss: 2.0023 - acc: 0.6860
...
242/242 [==============================] - 0s 125us/step - loss: 0.3685 - acc: 0.8430
Epoch 297/300
242/242 [==============================] - 0s 125us/step - loss: 0.3695 - acc: 0.8388
Epoch 298/300
242/242 [==============================] - 0s 121us/step - loss: 0.3844 - acc: 0.8347
Epoch 299/300
242/242 [==============================] - 0s 130us/step - loss: 0.3653 - acc: 0.8471
Epoch 300/300
242/242 [==============================] - 0s 142us/step - loss: 0.3721 - acc: 0.8471
```

Out[67]:
```
<keras.callbacks.History at 0x7f74eb3f4da0>
```

In [68]:
```
Y_pred_nn = model.predict(X_test)
```

In [69]:
```
Y_pred_nn.shape
```

Out[69]:
```
(61, 1)
```

In [70]:
```
rounded = [round(x[0]) for x in Y_pred_nn]

Y_pred_nn = rounded
```

In [71]:
```
score_nn = round(accuracy_score(Y_pred_nn,Y_test)*100,2)

print("The accuracy score achieved using Neural Network is: "+str(score_nn)+" %")

#Note: Accuracy of 85% can be achieved on the test set, by setting epochs=2000, and number of nodes = 11.
```

```
The accuracy score achieved using Neural Network is: 80.33 %
```

Fig 4.2.8 Neural Network accuracy

The efficiency of neural network is 80.33 %.

4.3 GRAPH

In the last section we have compared the accuracy score we have got from each graph.

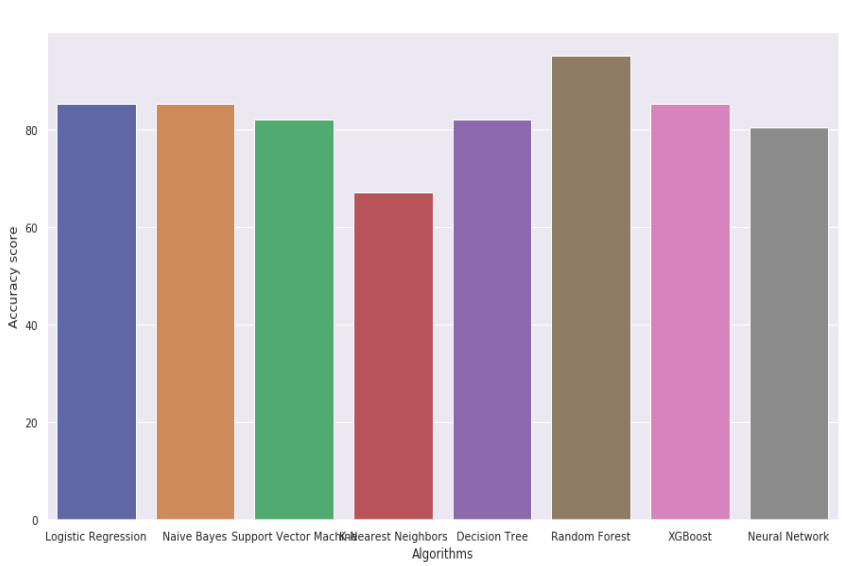

Fig 4.3.1 Different Algorithms accuracy score

The accuracy score achieved using Logistic Regression is: 85.25 %

The accuracy score achieved using Naive Bayes is: 85.25 %

The accuracy score achieved using Support Vector Machine is: 81.97 %

The accuracy score achieved using K-Nearest Neighbors is: 67.21 %

The accuracy score achieved using Decision Tree is: 81.97 %

The accuracy score achieved using Random Forest is: 95.08 %

The accuracy score achieved using XGBoost is: 85.25 %

The accuracy score achieved using Neural Network is: 80.33 %

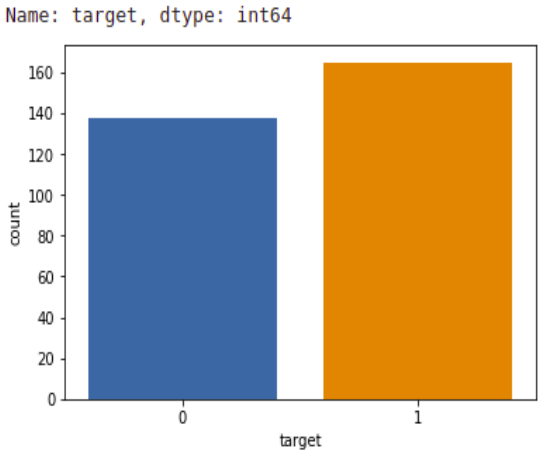

Fig 4.3.2 Describes the patients with heart problem (=0) and patients who have heart disease (=1)

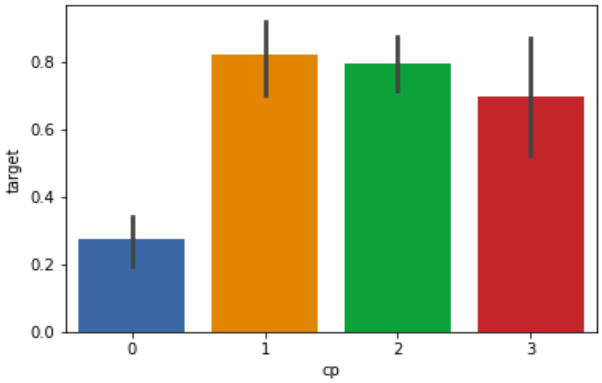

Fig 4.3.3 shows having chest pain=0 have least heart problem

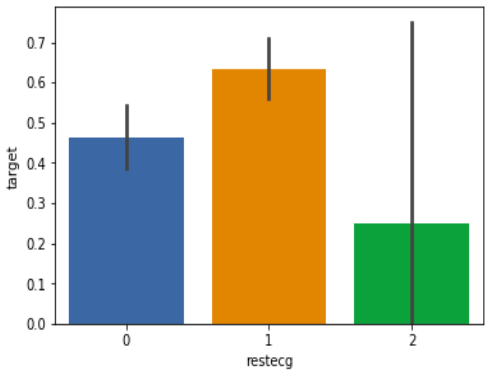

Fig 4.3.4 people with restecg '1' and '0' are much more likely to have a heart disease than with restecg '2'

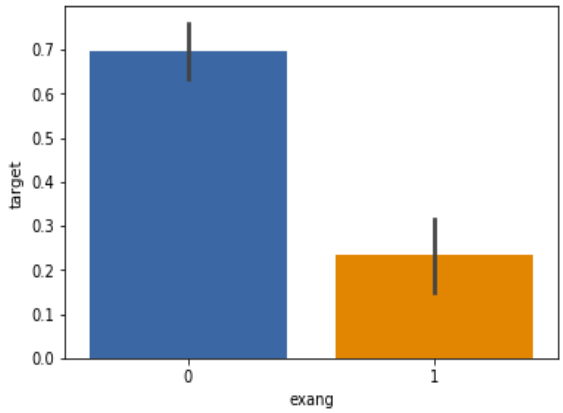

Fig 4.3.5 People with exang=1 i.e., Exercise induced angina are much less likely to have heart problems

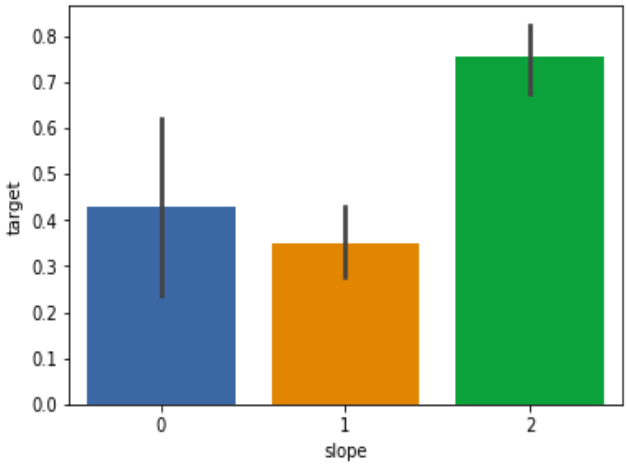

Fig 4.3.6 Slope '2' causes heart pain much more than Slope '0' and '1'

4.4 SNAPSHOT

I. Importing essential libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline

import os
print(os.listdir())

import warnings
warnings.filterwarnings('ignore')
```

```
['.ipynb_checkpoints', 'heart.csv', 'Heart_disease_prediction.ipynb', 'README.md']
```

Fig 4.4.1 libraries imported in the project.

II. Importing and understanding our dataset

In [2]:
```
dataset = pd.read_csv("heart.csv")
```

Verifying it as a 'dataframe' object in pandas

In [3]:
```
type(dataset)
```

Out[3]:
```
pandas.core.frame.DataFrame
```

Shape of dataset

In [4]:
```
dataset.shape
```

Out[4]:
```
(303, 14)
```

Printing out a few columns

In [5]:
```
dataset.head(5)
```

Out[5]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1

Fig 4.4.2 Description regarding Dataset

Logistic Regression

In [44]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()

lr.fit(X_train,Y_train)

Y_pred_lr = lr.predict(X_test)
```

In [45]:
```
Y_pred_lr.shape
```

Out[45]:
```
(61,)
```

In [46]:
```
score_lr = round(accuracy_score(Y_pred_lr,Y_test)*100,2)

print("The accuracy score achieved using Logistic Regression is: "+str(score_lr)+" %")
```

```
The accuracy score achieved using Logistic Regression is: 85.25 %
```

Fig 4.4.3 Logistic Regression Snapshot

Naive Bayes

In [47]:
```
from sklearn.naive_bayes import GaussianNB

nb = GaussianNB()

nb.fit(X_train,Y_train)

Y_pred_nb = nb.predict(X_test)
```

In [48]:
```
Y_pred_nb.shape
```

Out[48]:
```
(61,)
```

In [49]:
```
score_nb = round(accuracy_score(Y_pred_nb,Y_test)*100,2)

print("The accuracy score achieved using Naive Bayes is: "+str(score_nb)+" %")
```

```
The accuracy score achieved using Naive Bayes is: 85.25 %
```

Fig 4.4.4 Naïve Bayes Snapshot

SVM

In [50]:
```
from sklearn import svm

sv = svm.SVC(kernel='linear')

sv.fit(X_train, Y_train)

Y_pred_svm = sv.predict(X_test)
```

In [51]:
```
Y_pred_svm.shape
```

Out[51]:
```
(61,)
```

In [52]:
```
score_svm = round(accuracy_score(Y_pred_svm,Y_test)*100,2)

print("The accuracy score achieved using Linear SVM is: "+str(score_svm)+" %")
```

```
The accuracy score achieved using Linear SVM is: 81.97 %
```

Fig 4.4.5 SVM snapshot

K Nearest Neighbors

In [53]:
```
from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(X_train,Y_train)
Y_pred_knn=knn.predict(X_test)
```

In [54]:
```
Y_pred_knn.shape
```

Out[54]:
```
(61,)
```

In [55]:
```
score_knn = round(accuracy_score(Y_pred_knn,Y_test)*100,2)

print("The accuracy score achieved using KNN is: "+str(score_knn)+" %")
```

```
The accuracy score achieved using KNN is: 67.21 %
```

Fig 4.4.6 K Nearest Neighbours Snapshot

Decision Tree

In [56]:
```
from sklearn.tree import DecisionTreeClassifier

max_accuracy = 0


for x in range(200):
    dt = DecisionTreeClassifier(random_state=x)
    dt.fit(X_train,Y_train)
    Y_pred_dt = dt.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_dt,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)


dt = DecisionTreeClassifier(random_state=best_x)
dt.fit(X_train,Y_train)
Y_pred_dt = dt.predict(X_test)
```

In [57]:
```
print(Y_pred_dt.shape)
```

```
(61,)
```

In [58]:
```
score_dt = round(accuracy_score(Y_pred_dt,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_dt)+" %")
```

Fig 4.4.7 Decision Tree Snapshot

Random Forest

In [59]:
```
from sklearn.ensemble import RandomForestClassifier

max_accuracy = 0


for x in range(2000):
    rf = RandomForestClassifier(random_state=x)
    rf.fit(X_train,Y_train)
    Y_pred_rf = rf.predict(X_test)
    current_accuracy = round(accuracy_score(Y_pred_rf,Y_test)*100,2)
    if(current_accuracy>max_accuracy):
        max_accuracy = current_accuracy
        best_x = x

#print(max_accuracy)
#print(best_x)

rf = RandomForestClassifier(random_state=best_x)
rf.fit(X_train,Y_train)
Y_pred_rf = rf.predict(X_test)
```

In [60]:
```
Y_pred_rf.shape
```

Out[60]:
```
(61,)
```

In [61]:
```
score_rf = round(accuracy_score(Y_pred_rf,Y_test)*100,2)

print("The accuracy score achieved using Decision Tree is: "+str(score_rf)+" %")
```

Fig 4.4.8 Random Forest Snapshot

XGBoost

In [62]:
```
import xgboost as xgb

xgb_model = xgb.XGBClassifier(objective="binary:logistic", random_state=42)
xgb_model.fit(X_train, Y_train)

Y_pred_xgb = xgb_model.predict(X_test)
```

In [63]:
```
Y_pred_xgb.shape
```

Out[63]:
```
(61,)
```

In [64]:
```
score_xgb = round(accuracy_score(Y_pred_xgb,Y_test)*100,2)

print("The accuracy score achieved using XGBoost is: "+str(score_xgb)+" %")
```

```
The accuracy score achieved using XGBoost is: 85.25 %
```

Fig 4.4.9 XGBoost snapshot

Neural Network

In [65]:
```
from keras.models import Sequential
from keras.layers import Dense
```

```
Using TensorFlow backend.
```

In [66]:
```
# https://stats.stackexchange.com/a/136542 helped a lot in avoiding overfitting

model = Sequential()
model.add(Dense(11,activation='relu',input_dim=13))
model.add(Dense(1,activation='sigmoid'))

model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

In [67]:
```
model.fit(X_train,Y_train,epochs=300)
```

```
Epoch 1/300
242/242 [==============================] - 3s 10ms/step - loss: 4.7817 - acc: 0.6074
Epoch 2/300
242/242 [==============================] - 0s 123us/step - loss: 2.6633 - acc: 0.6033
Epoch 3/300
242/242 [==============================] - 0s 124us/step - loss: 2.3371 - acc: 0.6281
Epoch 4/300
242/242 [==============================] - 0s 127us/step - loss: 2.2938 - acc: 0.6488
Epoch 5/300
242/242 [==============================] - 0s 127us/step - loss: 2.0712 - acc: 0.6860
Epoch 6/300
242/242 [==============================] - 0s 125us/step - loss: 2.0645 - acc: 0.6653
Epoch 7/300
242/242 [==============================] - 0s 124us/step - loss: 2.0023 - acc: 0.6860
```

Fig 4.4.10 (i)

```
242/242 [==============================] - 0s 125us/step - loss: 0.3685 - acc: 0.8430
Epoch 297/300
242/242 [==============================] - 0s 125us/step - loss: 0.3695 - acc: 0.8388
Epoch 298/300
242/242 [==============================] - 0s 121us/step - loss: 0.3844 - acc: 0.8347
Epoch 299/300
242/242 [==============================] - 0s 130us/step - loss: 0.3653 - acc: 0.8471
Epoch 300/300
242/242 [==============================] - 0s 142us/step - loss: 0.3721 - acc: 0.8471
```

Out[67]:
```
<keras.callbacks.History at 0x7f74eb3f4da0>
```

In [68]:
```
Y_pred_nn = model.predict(X_test)
```

In [69]:
```
Y_pred_nn.shape
```

Out[69]:
```
(61, 1)
```

In [70]:
```
rounded = [round(x[0]) for x in Y_pred_nn]

Y_pred_nn = rounded
```

In [71]:
```
score_nn = round(accuracy_score(Y_pred_nn,Y_test)*100,2)

print("The accuracy score achieved using Neural Network is: "+str(score_nn)+" %")

#Note: Accuracy of 85% can be achieved on the test set, by setting epochs=2000, and number of nodes = 11.
```

```
The accuracy score achieved using Neural Network is: 80.33 %
```

Fig 4.4.10 (ii) Neural Network snapshot

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

We conclude that to build an accurate heart disease prediction model, a dataset with sufficient samples and correct data must be used. The dataset must be preprocessed accordingly because it is the most critical part to prepare the dataset to be used by the machine learning algorithm and get good results. Also, a suitable algorithm must be used when developing a prediction model. We can notice that Random Forest performed best, along with that, Artificial Neural Network (ANN) performed well in most models for predicting heart disease as well as Decision Tree (DT). Finally, the field of using machine learning for diagnosing heart disease is an important field, and it can help both healthcare professionals and patients. It is still a growing field, and despite the massive availability of patient data in hospitals or clinics, not much of it is published. Most researchers got their datasets from the same source which is the UCI repository. Since the quality of the dataset is an essential factor in the prediction's accuracy, more hospitals should be encouraged to publish high-quality datasets (while protecting the privacy of patients) so that researchers can have a good source to help them develop their models and obtain good results.

5.2 FUTURE WORK

There are many possible improvements that could be explored to improve the scalability and accuracy of this prediction system. As we have developed a generalized system, in future we can use this system for the analysis of different data sets. The performance of the health’s diagnosis can be improved significantly by handling numerous class labels in the prediction process, and it can be another positive direction of research. In DM warehouse, generally, the dimensionality of the heart database is high, so identification and selection of significant attributes for better diagnosis of heart disease are very challenging tasks for future research.

In future an intelligent system may be developed that can lead to selection of proper treatment methods for a patient diagnosed with heart disease.There are several treatment methods for a patient once diagnosed with a particular form of heart disease. Data mining can be of very good help in deciding the line of treatment to be followed by extracting knowledge from such suitable databases.

REFERENCES

[1] I. Kononenko, “Machine learning for medical diagnosis: History, state of the art and perspective,” Artificial Intelligence in Medicine, vol. 23, no. 1, pp. 89–109, 2001.

[2] J. Soni et al., “Intelligent and effective heart disease prediction system using weighted associative classifiers,” International Journal on Computer Science and Engineering, vol. 3, no. 6, pp. 2385–2392, 2011.

[3] N. Khateeb and M. Usman, “Efficient heart disease prediction system using k-nearest neighbor classification technique,” in Proceedings of the International Conference on Big Data and Internet of Thing (BDIOT), New York, NY, USA: ACM, 2017, pp. 21–26.

[4] H. Almarabeh and E. Amer, “A study of data mining techniques accuracy for healthcare,” International Journal of Computer Applications, vol. 168, no. 3, pp. 12–17, Jun 2017.

[5] M. Fatima and M. Pasha, “Survey of machine learning algorithms for disease diagnostic,” Journal of Intelligent Learning Systems and Applications, vol. 9, no. 01, pp. 1–16, 2017.

[6] S. Pouriyeh et al., “A comprehensive investigation and comparison of machine learning techniques in the domain of heart disease,” in Proceedings of IEEE Symposium on Computers and Communications (ISCC). Heraklion, Greece: IEEE, July 2017, pp. 204–207