

A PROJECT REPORT

ON

Disease Prediction using ML

For the partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

In

COMPUTER SCIENCE AND ENGINEERING

Submitted By

Abheesht Srivastava (1619210008)

Anshuman Bhatt (1619210046)

Rohan Sethia (1619210214)

Varun Agrawal (1619210291)

Under the Supervision of

Mr. Neeraj Chauhan



**G.L. BAJAJ INSTITUTE OF TECHNOLOGY &
MANAGEMENT, GREATER NOIDA**

Affiliated to



DR. APJ ABDUL KALAM TECHNICAL UNIVERSITY,

LUCKNOW

2020

Declaration

We hereby declare that the project work presented in this report entitled “**Disease Prediction using ML**” in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science & Engineering, submitted to A.P.J. Abdul Kalam Technical University, Lucknow, is based on my own work carried out at Department of Computer Science & Engineering, G.L. Bajaj Institute of Technology & Management, Greater Noida. The work contained in the report is original and project work reported in this report has not been submitted by me/us for award of any other degree or diploma.

Signature:

Name: Abheesht Srivastava

Roll No: 1619210008

Signature:

Name: Anshuman Bhatt

Roll No: 1619210046

Signature:

Name: Rohan Sethia

Roll No: 1619210214

Signature:

Name: Varun Agarwal

Roll No: 1619210291

Date:

Place: Greater Noida

Certificate

This is to certify that the Project report entitled “**“Disease Prediction using ML” done by Abheesht Srivastava (1619210008), Anshuman Bhatt (1619210046), Rohan Sethia (1619210214) and Varun Agrawal (1619210291)** is an original work carried out by them in Department of Computer Science & Engineering, G.L. Bajaj Institute of Technology & Management, Greater Noida under my guidance. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Date:

Mr. Neeraj Chauhan
Signature of the Supervisor

Dr. Sanjeev Kumar Pippal
Head of the Department

Acknowledgement

The merciful guidance bestowed to us by the almighty made us stick out this project to a successful end. We humbly pray with sincere heart for his guidance to continue forever.

We pay thanks to our project guide Mr. Neeraj Chauhan who has given guidance and light to us during this project. His/her versatile knowledge has cased us in the critical times during the span of this project.

We pay special thanks to our Head of Department Dr. Sanjeev Kumar Pippal who has been always present as a support and help us in all possible way during this project.

We also take this opportunity to express our gratitude to all those people who have been directly and indirectly with us during the completion of the project.

We want to thanks our friends who have always encouraged us during this project.

At the last but not least thanks to all the faculty of CSE department who provided valuable suggestions during the period of project.

Abstract

Over the last decade lack of disease prediction is the main reason for death in the world. Now due to progress in biomedical and healthcare communities, accurate study of medical data benefits early disease recognition, patient care and community services. When the quality of medical data is incomplete the exactness of study is reduced. Moreover, different regions exhibit unique appearances of certain regional diseases, which may results in weakening the prediction of disease outbreaks. In the proposed system, it provides machine learning algorithms for effective prediction of various disease occurrences in disease-frequent societies.

“Disease Prediction” system based on predictive modeling predicts the disease of the user on the basis of the symptoms that user provides as an input to the system. The system analyzes the symptoms provided by the user as input and gives the probability of the disease as an output

Disease Prediction is done by implementing three popular Decision Tree, Random Forest and Naïve Bayes Algorithm.

These Three Classifier algorithms calculates the probability of the disease. Compared to several typical estimate algorithms, the calculation exactness of our proposed algorithm reaches 94.8% with a convergence speed.

LIST OF FIGURES

Figure Number	Figure Name	Page No.
Figure 3.2.1(a)	Building a Naïve Bayes Classifier	15
Figure 3.2.1(b)	Naïve Bayes Algorithm Flowchart	17
Figure 3.2.1(c)	Bayes classifier and Naïve Bayes Classifier	18
Figure 3.2.1(d)	Gaussian Naïve Bayes	19
Figure 3.2.2.4(a)	Entropy Curve	28
Figure 3.2.2.4(b)	Entropy Formula	28
Figure 3.2.2.5(a)	Information Gain	30
Figure 3.2.2.5(b)	Gini Index	31
Figure 3.2.2.5(c)	Gain ratio	32
Figure 3.2.2.5(d)	Decision Tree Pruning	35
Figure 3.2.2.5(e)	Random Forest	37
Figure 3.2.2.5(f)	Decision Tree after pruning	42
Figure 3.2.3.1(a)	Decision Tree	45
Figure 3.2.3.2(a)	Random Forest Classifier	46
Figure 3.3(a)	GUI Screenshot	51
Figure 3.5(a)	State Diagram	56
Figure 4.4(a)	Sequence Diagram	57
Figure 4.4(b)	MapTrack in Testing Mode	81
	MapTrack in Testing Mode with Mock-Data	81

Table number	Table Name	Page No.
Table 2.1(a)	Predictive Accuracy of Bayes and other technique	9
Table 3.2.2.5(a)	Data Set	50

Candidate's Declaration.....	(ii)
Certificate	(iii)
Acknowledgement	(iv)
Abstract	(v)
List of Figures	(vii)
List of Tables	(viii)
Chapter 1. Introduction	1
1.1 Problem Definition	2
1.2 Project Overview	2
1.3. Scope	2
1.3.1 Widespread Usage	3
1.3.2 Software advancement	3
1.4 Hardware Specification	4
1.5 Software Requirement	4
1.5.1 Languages	4
1.5.2 Frontend	5
1.5.3 Data Storage and Stream	5
1.5.3.1 Data Storage	5
Chapter 2. Literature Survey	6
2.1 Literature Review	6
2.2 Goals and Objective	9
2.3 Existing System	9
2.4 Proposed System	9
2.5 Feasibility Study	10
2.5.1 Technical Feasibility	10
2.5.2 Economic Feasibility	10
2.5.3 Operational Feasibility	10
2.5.4 Legal Feasibility	11
2.5.5 Scheduling Feasibility	11
Chapter 3. System Analysis and Design	12
3.1 Application Overview/Methodology	12
3.2 Algorithm Implemented	13
3.2.1 Naïve Bayes	13
3.2.1.1 Introduction to Naïve Bayes	13

3.2.1.2	Naïve Bayes Classifier	14
3.2.1.3	Representation used by Naïve Bayes Model	14
3.2.1.4	Learn Naïve Bayes Model from Data	15
3.2.1.5	Calculating Class Probabilities	15
3.2.1.6	Calculating Conditional Probabilities	15
3.2.1.7	Make Predictions with Naive Bayes Model	18
3.2.1.8	Gaussian Naïve Bayes	19
3.2.1.9	Representation for Gaussian Naïve Bayes	19
3.2.1.10	Learn Gaussian Naïve Bayes model from Data	19
3.2.1.11	Make Predictions with a Gaussian Naïve Bayes Model	20
3.2.1.12	Best Prepare your Data for Naïve Bayes	21
3.2.1.13	Naïve Bayes using Numpy and Pandas	21
3.2.2	Decision Tree	22
3.2.2.1	Types of Decision Tree	23
3.2.2.2	Creating Decision Tree	24
3.2.2.3	Working of a Decision Tree	25
3.2.2.4	Entropy	27
3.2.2.5	Information Gain	29
3.2.2.6	Optimization of Decision Tree Classifier	41
3.2.2.7	Decision Tree using numpy and panda	43
3.2.3	Random Forest	44
3.2.3.1	Decision Tree	44
3.2.3.2	Random Forest Classifier	46
3.2.3.3	Random Forest in python	47
3.2.3.4	Random Forest	49
3.2.3.5	Random Forest in Practice	49
3.3	Display	50
3.4	State Diagram	56
3.5	Sequence Diagram	57

4.1.	Type of testing	68
4.1.1	Unit testing	68
4.1.2	Functional testing	69
4.1.3	Integration testing	69
4.2	The Tools	69
4.3	Why Testing	70
Chapter 5.	Conclusion & Future Scope.....	72
References		

Chapter 1

Introduction

With the advance of technology, analytics equipment, more devotion has been paid to disease expectation from the perception of big data inquiry, various explores have been conducted by choosing the features mechanically from a large number of data to improve the truth of menace classification rather than the formerly selected physiognomies. However, those prevailing work mostly measured structured data. Thus, risk organization based on prediction and analysis, the following tasks remain: How should the mislaid data be lectured? How should the main chronic diseases in a positive county and the main faces of the disease in the region be gritty? How can prediction and analysis expertise be used to estimate the disease and generate a better method? At present, when one suffers from particular disease, then the person has to visit to doctor which is time consuming and costly too. Also if the user is out of reach of doctor and hospitals it may be difficult for the user as the disease cannot be identified. So, if the above process can be completed using an automated program which can save time as well as money, it could be easier to the patient which can make the process easier. There are other related Disease Prediction System using data mining techniques that analyzes the risk level of the patient. Disease Predictor is a web based application that predicts the disease of the user with respect to the symptoms given by the user. It is preloaded with data sets collected from different health related sites. With the help of Disease Predictor the user will be able to know the probability of the disease with the given symptoms. As the use of internet is growing every day, people are always curious to know different new things. People always try to refer to the internet if any problem arises. People have access to internet than hospitals and doctors. People do not have immediate option when they suffer with particular disease. So, this system can be helpful to the people as they have access to internet 24 hours.

1.1 Problem Definition

There are many tools related to disease prediction. But particularly heart related diseases have been analyzed and risk level is generated. But generally there are no such tools that are used for prediction of general diseases. So Disease Predictor helps for the prediction of the general diseases. Machine can predict diseases but cannot predict the sub types of the diseases caused by occurrence of one disease. It fails to predict all possible conditions of the people. Existing system handles only structured data. The prediction system are broad and ambiguous. In current past, countless disease estimate classifications have been advanced and in procedure. The standing organizations arrange a blend of machine learning algorithms which are judiciously exact in envisaging diseases. However the restraint with the prevailing systems are speckled. First, the prevailing systems are dearer only rich people could pay for to such calculation systems. And also, when it comes to folks, it becomes even higher. Second, the guess systems are non-specific and indefinite so far. So that, a machine can envisage a positive disease but cannot expect the sub types of the diseases and diseases caused by the existence of one bug.

1.2 Project Overview

Disease Predictor is built as a generic platform to solve the problem of prediction and analysis of Disease. It is a centralized application/service that can be used by citizens to keep themselves aware of the disease and casualties if any.

To solve these problems, it see the structured and unstructured data in healthcare field to assess the risk of disease. Disease can be predicted anytime. The system uses Decision tree map algorithm, Random Forest Algorithm, and Naïve Bayes algorithm to generate the pattern and causes of disease. It clearly shows the diseases and sub diseases.

1.3 Scope

The system has been implemented with the outmost accuracy on the free dataset of patient data. The current system covers only the general diseases or the more commonly occurring disease, the plan is to include disease of higher fatality, like various cancers in future, so that early prediction and treatment could be done, and the fatality rate of deadly diseases like cancer decreases, with the economic benefit in long sight as well. The tool is so designed as to help family doctors and healthcare workers identify people with possible diagnosis of early cancer,

heart disease, arthritis. This tool was thoroughly tested to ensure that it is consistent and valid. It is hoped that the tool will go ahead to earlier detection of diseases.

1.3.1 Widespread usage

Due to the increase in number of deaths due to lack of accurate prediction and lack of medical facilities. Patients consume widespread and inappropriate use of antibiotics in the hospital. There is also evidence that the widespread use of such drugs are leading to enormous amount of untimely deaths. This disease predictor tool calculate disease risk as well as interpret the results to encourage widespread use of this tool. This tool is very convenient and an effective measure to monitor and track the diseases, patients are suffering from. Companies now are using this tool to gather information about the medical history and data of their employees. This tool is able to generate report and store on the system of user. This saves the memory of server and thus can be run on a low cost server. This advantage of hoisting it on low cost server also adds in its popularity and widespread usage of disease predictor tool.

1.3.2 Software advancement

Significant advances in information technology results in excessive growth of data in health care informatics. Health care informatics data includes hospital details, patient's details, disease details and treatment cost. These huge data are generated from different sources and format. It can have irrelevant attributes and missing data. Applying data mining techniques is a key approach to extract knowledge from large disease data. Data mining has various methods to extract knowledge from huge disease data set. Data mining techniques like classification, clustering and rule mining can be used to analyze data and extract meaningful information. Some of the important current applications of data mining in health care includes predicting the future outcomes of diseases based on previous data collected from similar diseases, diagnosis of disease based on patient data, analyzing treatment costs and demand of resources, preprocessing of noisy, missing data and minimizing the time to wait for the disease diagnosis. New and current data mining tools and technologies are used in disease diagnosis and health care informatics to improve the health care services in cost effective manner and minimizing the time for disease diagnosis.

1.4 Hardware Specification

1. Laptop or Work area to use the application. Disease Prediction perceives the different diseases given as input by user. The system further calculates the results and gives the output on the screen to the user as the user clicks on the button of all the three algorithms. The tool is able to generate report in pdf format and store it on the system of user.
2. 4GB of reserve memory was held all through testing, and the tool required no extra memory to work on. The tool was running seamlessly on the provided memory.
3. 1.4 GHz Processor is enough to run this prediction tool over the system. This configuration system is able to handle user requests efficiently.

1.5 Software requirements

1.5.1 Languages

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented, and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Python was conceived in the late 1980s as a successor to the ABC language. Python 2.0, released in 2000, introduced features like list comprehensions and a garbage collection system with reference counting.

Python 3.0, released in 2008, was a major revision of the language that is not completely backward-compatible, and much Python 2 code does not run unmodified on Python 3. Python interpreters are available for many operating systems.

The language's core philosophy is summarized in the document The Zen of Python (PEP 20), which includes aphorisms such as:

- Beautiful is better than ugly.

- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Readability counts.

Rather than having all of its functionality built into its core, Python was designed to be highly extensible.

1.5.2 Frontend

Graphical User Interface is Developed over Tkinter is a Python binding to the Tk GUI toolkit. It is the standard Python interface to the Tk GUI toolkit, and is Python's de facto standard GUI. Tkinter is included with standard Linux, Microsoft Windows and Mac OS X installs of Python.

As with most other modern Tk bindings, Tkinter is implemented as a Python wrapper around a complete Tcl interpreter embedded in the Python interpreter. Tkinter calls are translated into Tcl commands which are fed to this embedded interpreter, thus making it possible to mix Python and Tcl in a single application.

There are several popular GUI library alternatives available

1.5.3 Data Storage and Stream

1.5.3.1 Data Storage

As the tool is able to generate the medical report of the user and uses a data storage media to store it. This tool is able to handle and store the report generated of user after the action of report generation is fired. The application stores the report on the local storage of the system on which it is running. This saves the memory from being over utilized and thus making the server slow to operate. The report is saved on the secondary memory of the system which is most of the times a hard disk.

Chapter 2

Literature Survey

2.1 Literature Review

K.M. Al-Aidaroos, A.A. Bakar and Z. Othman have conducted the research for the best medical diagnosis mining technique. For this authors compared Naïve Baeyes with five other classifiers i.e. Logistic Regression (LR), KStar (K*), Decision Tree (DT), Neural Network (NN) and a simple rule-based algorithm (ZeroR). For this, 15 real-world medical problems from the UCI machine learning repository (Asuncion and Newman, 2007) were selected for evaluating the performance of all algorithms. In the experiment it was found that NB outperforms the other algorithms in 8 out of 15 data sets so it was concluded that the predictive accuracy results in Naïve Baeyes is better than other techniques.

Table 1- Predictive Accuracy of Bayes and other Technique

Medical Problems	NB	LR	K*	DT	NN	ZeroR
Breast Cancer wise	97.3	92.98	95.72	94.57	95.57	65.52
Breast Cancer	72.7	67.77	73.73	74.28	66.95	70.3
Dermatology	97.43	96.89	94.51	94.1	96.45	30.6
Echocardiogram	95.77	94.59	89.38	96.41	93.64	67.86
Liver Disorders	54.89	68.72	66.82	65.84	68.73	57.98
Pima Diabetes	75.75	77.47	70.19	74.49	74.75	65.11
Haeberman	75.36	74.41	73.73	72.16	70.32	73.53
Heart-c	83.34	83.7	75.18	77.13	80.99	54.45
Heart-statlog	84.85	84.04	73.89	75.59	81.78	55.56
Heart-b	83.95	84.23	77.83	80.22	80.07	63.95
Hepatitis	83.81	83.89	80.17	79.22	80.78	79.38
Lung Cancer	53.25	47.25	41.67	40.83	44.08	40
Lymphpgraphy	84.97	78.45	83.18	78.21	81.81	54.76
Postooerative Patient	68.11	61.11	61.67	69.78	58.54	71.11
Primary tumor	49.71	41.62	38.02	41.39	40.38	24.78
Wins	8\15	5\15	0\15	2\15	1\15	1\15

(Al-Aidaroos, Bakar, & Othman, 2012)

Darcy A. Davis, Nitesh V. Chawla, NicholasBlumm, Nicholas Christakis, Albert-Laszlo Barabasi have found that global treatment of chronic disease is neither time or cost efficient.

So the authors conducted this research to predict future disease risk. For this CARE was used (which relies only on a patient's medical history using ICD- 9-CM codes in order to predict future diseases risks). CARE combines collaborative filtering methods with clustering to predict each patient's greatest disease risks based on their own medical history and that of similar patients. Authors have also described an Iterative version, ICARE, which incorporates ensemble concepts for improved performance. These novel systems require no specialized information and provide predictions for medical conditions of all kinds in a single run. The impressive future disease coverage of ICARE represents more accurate early warnings for thousands of diseases, some even years in advance. Applied to full potential, the CARE framework can be used explore a broader disease histories, suggest previously unconsidered concerns, and facilitating discussion about early testing and prevention.

(A.Davis, V.Chawla, Blumm, Christakis, & Barbasi, 2008)

JyotiSoni, Ujma Ansari, Dipesh Sharma and SunitaSoni have done this research paper intoprovide a survey of current techniques of knowledge discovery in databases using data mining techniques that are in use in today's medical research particularly in Heart Disease Prediction. Number of experiment has been conducted to compare the performance of predictive data mining technique on the same dataset and the outcome reveals that Decision Tree outperforms and sometime Bayesian classification is having similar accuracy as of decision tree but other predictive methods like KNN, Neural Networks, Classification based on clustering is not performing well.

(JyotiSoni, Ansari, Sharma, & Soni, 2011)

Shadab Adam Pattekari and AsmaParveen have conducted a research using Naïve Bayes Algorithm to predict the heart diseases where user provides the data which is compared with trained set of values. So from this research, patients were able to provide their basic information which is compared with the data and the heart disease is predicted.

(Adam & Parveen, 2012)

M.A.NisharaBanu, B Gomathy used medical data mining techniques like association rule mining, classification, clustering I to analyze the different kinds of heart based problems. Decision tree is made to illustrate every possible outcome of a decision. Different rules are made to get the best outcome. In this research age, sex, smoking, overweight, alcohol intake,

blood sugar, hear rate, blood pressure are the parameters used for making the decisions. Risk level for different parameters are stored with their id's ranging (1-8). ID lesser than of 1 of weight contains the normal level of prediction and higher ID other than 1 comprise the higher risk levels .K-means clustering technique is used to study the pattern in the dataset. The algorithm clusters information into k groups. Each point in the dataset is assigned to the closed cluster. Each cluster center is recomputed as the average of the points in that cluster.

(NisharBanu, MA; Gomathy, 2013)

2.2 Goals and Objective

The aimed to build a fully functional system in order to achieve an efficiency in faster health treatment and online consultations system. The overall mission of system development is to make the primary treatment quickly and easily complete online consultation system.

2.3 Existing System

Prediction using traditional disease risk model usually involves a machine learning and supervised learning algorithm which uses training data with the labels for the training of the models. High-risk and Low-risk patient classification is done in groups test sets. But these models are only valuable in clinical situations and are widely studied. A system for sustainable health monitoring using smart clothing by Chen et.al. He thoroughly studied heterogeneous systems and was able to achieve the best results for cost minimization on the tree and simple path cases for heterogeneous systems.

Existing System Prediction using traditional disease risk model usually involves a machine learning and supervised learning algorithm which uses training data with the labels for the

training of the models. High-risk and Low-risk patient classification is done in groups test sets. But these models are only valuable in clinical situations and are widely studied. A system for sustainable health monitoring using smart clothing by Chen et.al. He thoroughly studied heterogeneous systems and was able to achieve the best results for cost minimization on the tree and simple path cases for heterogeneous systems.

2.4 Proposed System

The system is designed to use intelligent data mining techniques to guess the most accurate illness based on patient's symptoms. If user's symptoms do not exactly match any disease in the database, then it shows the diseases user could probably have based on his/her symptoms. It also generates the reports of the patients

2.5 Feasibility Study

2.5.1 Technical Feasibility

A **technical feasibility** study assesses the details of how you intend to deliver a product or service to customers. Think materials, labor, transportation, where your business will be located, and the technology that will be necessary to bring all this together.

The project is technically feasible as it can be built using the existing available technologies. It is an applications that uses Python. The technology required by Disease Predictor is available and hence it is technically feasible.

2.5.2 Economic Feasibility

The purpose of an economic feasibility study (EFS) is to demonstrate the net benefit of a proposed project for accepting or disbursing electronic funds/benefits, taking into consideration the benefits and costs to the agency, other state agencies, and the general public as a whole.

The project is economically feasible as the cost of the project is involved only in the hosting of the project. As the data samples increases, which consume more time and processing power. In that case better processor might be needed.

2.5.3 Operational Feasibility

Operational feasibility is the measure of how well a proposed system solves the problems, and takes advantage of the opportunities identified during scope definition and how it satisfies the requirements identified in the requirements analysis phase of system development.

The project is operationally feasible as the user having basic knowledge about computer and Internet. Disease Predictor is based on client-server architecture where client is users and server is the machine where datasets are stored.

2.5.4 Legal Feasibility

Legal feasibility is the study to know if the proposed project conform the legal and ethical requirements. This project conforms all the legal and ethical requirement.

2.5.5 Scheduling Feasibility

An exceptionally critical piece of plausibility examine is booking Practicality. It is additionally assume a significant jobs to finish the venture in its timetable time .An undertaking some time not be ineffective in the event that it isn't done in its limited time allotment. Here we may foresee the time prerequisite to finish different assignment of the whole venture.

Chapter 3

SYSTEM ANALYSIS & DESIGN

3.1 Application Overview/Methodology

Guide Track utilizes different sorts of the innovation to actualize the proposed work. The primary feature about the application is the innovation. Utilizes innovation which is a free and open-source programming stack for structure dynamic sites and web applications

The stack is Python using Numpy, Pandas, and Tkinter libraries for development. Since all parts of the MERN stack bolster programs that are written in JavaScript, MERN applications can be written in one language for both server-side and customer side execution situations.

3.2 Algorithms Implemented

3.2.1 Naive Bayes

3.2.1.1 Introduction to Naive Bayes

In machine learning we are often interested in selecting the best hypothesis (h) given data (d).

In a classification problem, our hypothesis (h) may be the class to assign for a new data instance (d).

One of the easiest ways of selecting the most probable hypothesis given the data that we have that we can use as our prior knowledge about the problem. Bayes' Theorem provides a way that we can calculate the probability of a hypothesis given our prior knowledge.

Bayes' Theorem is stated as:

$$P(h|d) = (P(d|h) * P(h)) / P(d)$$

Where

- **P(h|d)** is the probability of hypothesis h given the data d. This is called the posterior probability.
- **P(d|h)** is the probability of data d given that the hypothesis h was true.
- **P(h)** is the probability of hypothesis h being true (regardless of the data). This is called the prior probability of h.
- **P(d)** is the probability of the data (regardless of the hypothesis).

You can see that we are interested in calculating the posterior probability of P(h|d) from the prior probability p(h) with P(D) and P(d|h).

After calculating the posterior probability for a number of different hypotheses, you can select the hypothesis with the highest probability. This is the maximum probable hypothesis and may formally be called the maximum a posteriori (MAP) hypothesis.

This can be written as:

$$MAP(h) = \max(P(h|d))$$

or

$$MAP(h) = \max((P(d|h) * P(h)) / P(d))$$

or

$$MAP(h) = \max(P(d|h) * P(h))$$

The P(d) is a normalizing term which allows us to calculate the probability. We can drop it when we are interested in the most probable hypothesis as it is constant and only used to normalize.

Back to classification, if we have an even number of instances in each class in our training data, then the probability of each class (e.g. P(h)) will be equal. Again, this would be a constant term in our equation and we could drop it so that we end up with:

$$MAP(h) = \max(P(d|h))$$

This is a useful exercise, because when reading up further on Naive Bayes you may see all of these forms of the theorem.

3.2.1.2 Naive Bayes Classifier

Naive Bayes is a classification algorithm for binary (two-class) and multi-class classification problems. The technique is easiest to understand when described using binary or categorical input values.

It is called *naive Bayes* or *idiot Bayes* because the calculation of the probabilities for each hypothesis are simplified to make their calculation tractable. Rather than attempting to calculate the values of each attribute value $P(d_1, d_2, d_3|h)$, they are assumed to be conditionally independent given the target value and calculated as $P(d_1|h) * P(d_2|H)$ and so on.

This is a very strong assumption that is most unlikely in real data, i.e. that the attributes do not interact. Nevertheless, the approach performs surprisingly well on data where this assumption does not hold.

3.2.1.3 Representation Used By Naive Bayes Models

The representation for naive Bayes is probabilities.

A list of probabilities are stored to file for a learned naive Bayes model. This includes:

- **Class Probabilities:** The probabilities of each class in the training dataset.
- **Conditional Probabilities:** The conditional probabilities of each input value given each class value.

3.2.1.4 Learn a Naive Bayes Model from Data

Learning a naive Bayes model from your training data is fast.

Training is fast because only the probability of each class and the probability of each class given different input (x) values need to be calculated. No coefficients need to be fitted by optimization procedures.

Building a Naive Bayes classifier

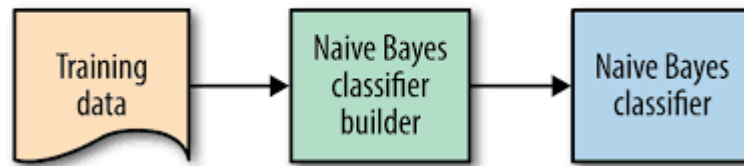


Figure 3.2.1(a) Building a Naïve Bayes Classifier

3.2.1.5 Calculating Class Probabilities

The class probabilities are simply the frequency of instances that belong to each class divided by the total number of instances.

For example in a binary classification the probability of an instance belonging to class 1 would be calculated as:

$$P(\text{class}=1) = \text{count}(\text{class}=1) / (\text{count}(\text{class}=0) + \text{count}(\text{class}=1))$$

In the simplest case each class would have the probability of 0.5 or 50% for a binary classification problem with the same number of instances in each class.

3.2.1.6 Calculating Conditional Probabilities

The conditional probabilities are the frequency of each attribute value for a given class value divided by the frequency of instances with that class value.

For example, if a “*weather*” attribute had the values “*sunny*” and “*rainy*” and the class attribute had the class values “*go-out*” and “*stay-home*“, then the conditional probabilities of each weather value for each class value could be calculated as:

- $P(\text{weather}=\text{sunny}|\text{class}=\text{go-out}) = \text{count}(\text{instances with weather}=\text{sunny and class}=\text{go-out}) / \text{count}(\text{instances with class}=\text{go-out})$
- $P(\text{weather}=\text{sunny}|\text{class}=\text{stay-home}) = \text{count}(\text{instances with weather}=\text{sunny and class}=\text{stay-home}) / \text{count}(\text{instances with class}=\text{stay-home})$
- $P(\text{weather}=\text{rainy}|\text{class}=\text{go-out}) = \text{count}(\text{instances with weather}=\text{rainy and class}=\text{go-out}) / \text{count}(\text{instances with class}=\text{go-out})$
- $P(\text{weather}=\text{rainy}|\text{class}=\text{stay-home}) = \text{count}(\text{instances with weather}=\text{rainy and class}=\text{stay-home}) / \text{count}(\text{instances with class}=\text{stay-home})$

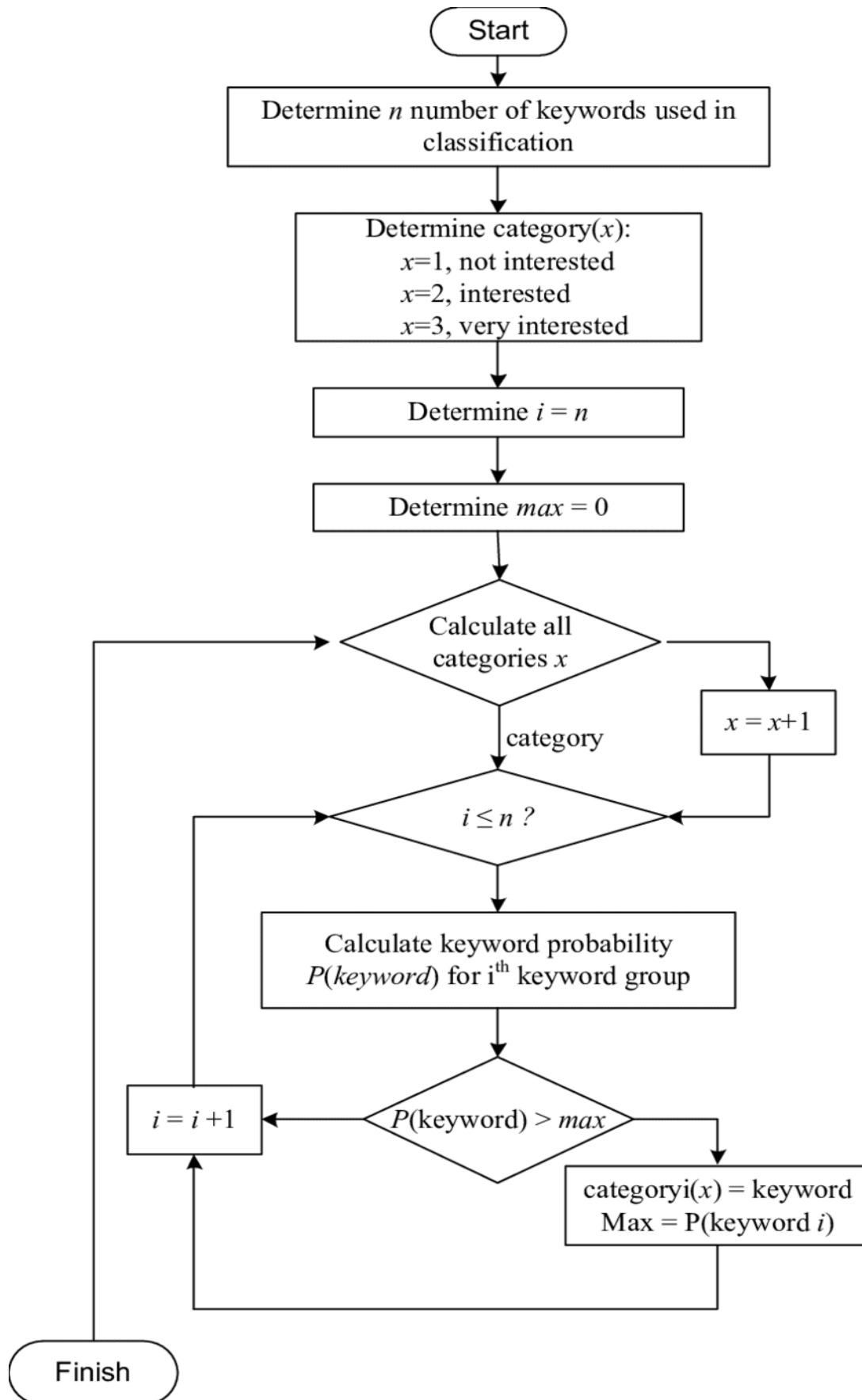


Figure 3.2.1(b) Naïve Bayes Algorithm Flowchart

3.2.1.7 Make Predictions with a Naive Bayes Model

Given a naive Bayes model, you can make predictions for new data using Bayes theorem.

$$\text{MAP} (h) = \max (P (d|h) * P (h))$$

Using our example above, if we had a new instance with the *weather* of *sunny*, we can calculate:

$$\text{Go-out} = P (\text{weather}=\text{sunny} \text{ class}=\text{go-out}) * P (\text{class}=\text{go-out})$$

$$\text{stay-home} = P (\text{weather}=\text{sunny} \text{ class}=\text{stay-home}) * P (\text{class}=\text{stay-home})$$

We can choose the class that has the largest calculated value. We can turn these values into probabilities by normalizing them as follows:

$$P (\text{go-out}|\text{weather}=\text{sunny}) = \text{go-out} / (\text{go-out} + \text{stay-home})$$

$$P (\text{stay-home}|\text{weather}=\text{sunny}) = \text{stay-home} / (\text{go-out} + \text{stay-home})$$

If we had more input variables we could extend the above example. For example, pretend we have a “*car*” attribute with the values “*working*” and “*broken*“. We can multiply this probability into the equation.

For example below is the calculation for the “go-out” class label with the addition of the car input variable set to “working”:

$$\text{go-out} = P(\text{weather}=\text{sunny} \text{ class}=\text{go-out}) * P(\text{car}=\text{working}|\text{class}=\text{go-out}) * P(\text{class}=\text{go-out})$$

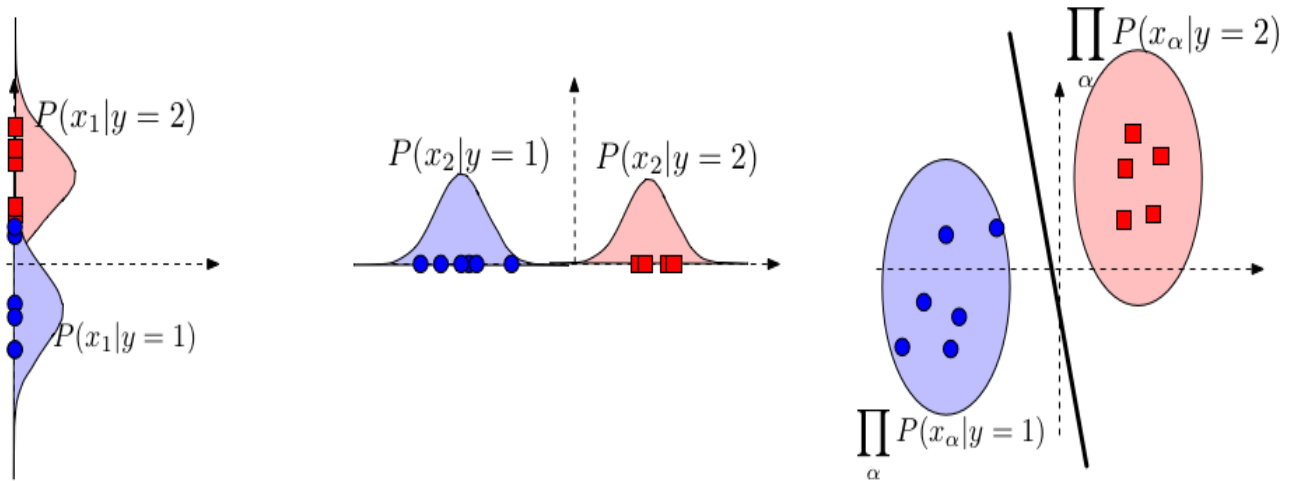


Figure 3.2.1(c) Bayes classifier and Naïve Bayes Classifier

3.2.1.8 Gaussian Naive Bayes

Naive Bayes can be extended to real-valued attributes, most commonly by assuming a Gaussian distribution.

This extension of naive Bayes is called Gaussian Naive Bayes. Other functions can be used to estimate the distribution of the data, but the Gaussian (or Normal distribution) is the easiest to work with because you only need to estimate the mean and the standard deviation from your training data.

3.2.1.9 Representation for Gaussian Naive Bayes

Above, we calculated the probabilities for input values for each class using a frequency. With real-valued inputs, we can calculate the mean and standard deviation of input values (x) for each class to summarize the distribution.

This means that in addition to the probabilities for each class, we must also store the mean and standard deviations for each input variable for each class.

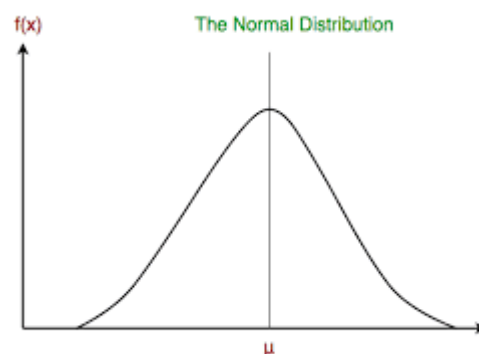


Figure 3.2.1(d) Gaussian Naïve Bayes

3.2.1.10 Learn a Gaussian Naive Bayes Model from Data

This is as simple as calculating the mean and standard deviation values of each input variable (x) for each class value.

$$\text{Mean}(x) = 1/n * \text{sum}(x)$$

Where n is the number of instances and x are the values for an input variable in your training data.

We can calculate the standard deviation using the following equation:

Standard deviation(x) = sqrt (1/n * sum (xi-mean(x) ^2))

This is the square root of the average squared difference of each value of x from the mean value of x, where n is the number of instances, sqrt () is the square root function, sum () is the sum function, xi is a specific value of the x variable for the i'th instance and mean(x) is described above, and ^2 is the square.

3.2.1.11 Make Predictions with a Gaussian Naive Bayes Model

Probabilities of new x values are calculated using the Gaussian Probability Density Function (PDF).

When making predictions these parameters can be plugged into the Gaussian PDF with a new input for the variable, and in return the Gaussian PDF will provide an estimate of the probability of that new input value for that class.

$$\text{Pdf}(x, \text{mean}, \text{sd}) = (1 / (\text{sqrt}(2 * \text{PI}) * \text{sd})) * \exp(-((x-\text{mean})^2)/(2*\text{sd}^2)))$$

Where pdf(x) is the Gaussian PDF, sqrt() is the square root, mean and sd are the mean and standard deviation calculated above, PI is the numerical constant, exp() is the numerical constant e or Euler's number raised to power and x is the input value for the input variable.

We can then plug in the probabilities into the equation above to make predictions with real-valued inputs.

For example, adapting one of the above calculations with numerical values for weather and car: go-out = P(pdf(weather)|class=go-out) * P(pdf(car)|class=go-out) * P(class=go-out)

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

3.2.1.12 Best Prepare Your Data For Naive Bayes

- **Categorical Inputs:** Naive Bayes assumes label attributes such as binary, categorical or nominal.
- **Gaussian Inputs:** If the input variables are real-valued, a Gaussian distribution is assumed. In which case the algorithm will perform better if the univariate distributions of your data are Gaussian or near-Gaussian. This may require removing outliers (e.g. values that are more than 3 or 4 standard deviations from the mean).
- **Classification Problems:** Naive Bayes is a classification algorithm suitable for binary and multiclass classification.
- **Log Probabilities:** The calculation of the likelihood of different class values involves multiplying a lot of small numbers together. This can lead to an underflow of numerical precision. As such it is good practice to use a log transform of the probabilities to avoid this underflow.
- **Kernel Functions:** Rather than assuming a Gaussian distribution for numerical input values, more complex distributions can be used such as a variety of kernel density functions.
- **Update Probabilities:** When new data becomes available, you can simply update the probabilities of your model. This can be helpful if the data changes frequently.

3.2.1.13 Naive Bayes using Numpy and Pandas

```
from tkinter import *
import numpy as np
import pandas as pd
def NaiveBayes():
    from sklearn.naive_bayes import GaussianNB
    gnb = GaussianNB()
    gnb=gnb.fit(X,np.ravel(y))

    # calculating accuracy-----
    from sklearn.metrics import accuracy_score
    y_pred=gnb.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred,normalize=False))
```

```
# -----
```

```
psymptoms
```

=

```
[Symptom1.get(),Symptom2.get(),Symptom3.get(),Symptom4.get(),Symptom5.get()]
```

```
for k in range(0,len(l1)):
```

```
    for z in psymptoms:
```

```
        if(z==l1[k]):
```

```
            l2[k]=1
```

```
inputtest = [l2]
```

```
predict = gnb.predict(inputtest)
```

```
predicted=predict[0]
```

```
h='no'
```

```
for a in range(0,len(disease)):
```

```
    if(predicted == a):
```

```
        h='yes'
```

```
        break
```

```
if (h=='yes'):
```

```
    t3.delete("1.0", END)
```

```
    t3.insert(END, disease[a])
```

```
else:
```

```
    t3.delete("1.0", END)
```

```
    t3.insert(END, "Not Found")
```

3.2.2 Decision Tree

Decision Tree algorithm belongs to the family of supervised learning algorithms. Unlike other supervised learning algorithms, the decision tree algorithm can be used for solving regression and classification problems too.

The goal of using a Decision Tree is to create a training model that can use to predict the class or value of the target variable by learning simple decision rules inferred from prior data (training data).

In Decision Trees, for predicting a class label for a record we start from the root of the tree. We compare the values of the root attribute with the record's attribute. On the basis of comparison, we follow the branch corresponding to that value and jump to the next node.

3.2.2.1 Types of Decision Trees

Types of decision trees are based on the type of target variable we have. It can be of two types:

1. **Categorical Variable Decision Tree:** Decision Tree which has a categorical target variable then it called a Categorical variable decision tree.
2. **Continuous Variable Decision Tree:** Decision Tree has a continuous target variable then it is called Continuous Variable Decision Tree.

Example: Let's say we have a problem to predict whether a customer will pay his renewal premium with an insurance company (yes/ no). Here we know that the income of customers is a significant variable but the insurance company does not have income details for all customers. Now, as we know this is an important variable, then we can build a decision tree to predict customer income based on occupation, product, and various other variables. In this case, we are predicting values for the continuous variables.

Important Terminology related to Decision Trees

1. Root Node: It represents the entire population or sample and this further gets divided into two or more homogeneous sets.

2. Splitting: It is a process of dividing a node into two or more sub-nodes.

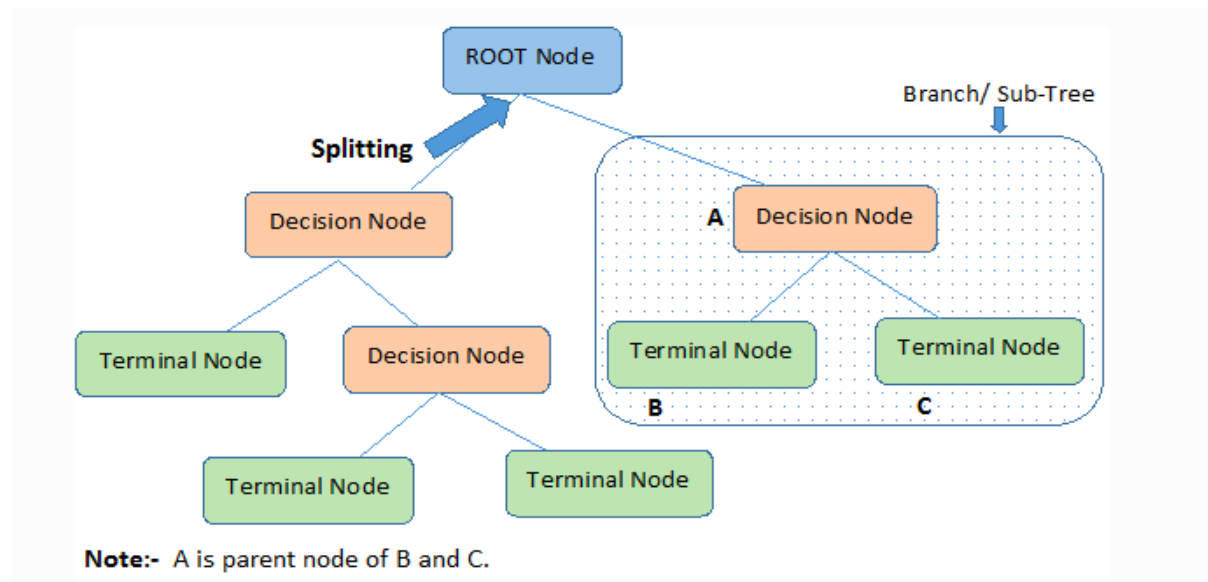
3. Decision Node: When a sub-node splits into further sub-nodes, then it is called the decision node.

4. Leaf / Terminal Node: Nodes do not split is called Leaf or Terminal node.

5. Pruning: When we remove sub-nodes of a decision node, this process is called pruning. You can say the opposite process of splitting.

6. Branch / Sub-Tree: A subsection of the entire tree is called branch or sub-tree.

7. Parent and Child Node: A node, which is divided into sub-nodes is called a parent node of sub-nodes whereas sub-nodes are the child of a parent node.



Decision trees classify the examples by sorting them down the tree from the root to some leaf/terminal node, with the leaf/terminal node providing the classification of the example.

Each node in the tree acts as a test case for some attribute, and each edge descending from the node corresponds to the possible answers to the test case. This process is recursive in nature and is repeated for every subtree rooted at the new node.

3.2.2.2 Creating Decision Tree

Below are some of the assumptions we make while using Decision tree:

- In the beginning, the whole training set is considered as the **root**.
- Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
- Records are distributed recursively on the basis of attribute values.
- Order to placing attributes as root or internal node of the tree is done by using some statistical approach.

Decision Trees follow Sum of Product (SOP) representation. The Sum of product (SOP) is also known as Disjunctive Normal Form. For a class, every branch from the root of the tree to a leaf node having the same class is conjunction (product) of values, different branches ending in that class form a disjunction (sum).

The primary challenge in the decision tree implementation is to identify which attributes do we need to consider as the root node and each level. Handling this is known as the attributes selection. We have different attributes selection measures to identify the attribute which can be considered as the root node at each level.

3.2.2.4 Working of Decision Tree

The decision of making strategic splits heavily affects a tree's accuracy. The decision criteria are different for classification and regression trees.

Decision trees use multiple algorithms to decide to split a node into two or more sub-nodes. The creation of sub-nodes increases the homogeneity of resultant sub-nodes. In other words, we can say that the purity of the node increases with respect to the target variable. The decision tree splits the nodes on all available variables and then selects the split which results in most homogeneous sub-nodes.

The algorithm selection is also based on the type of target variables. Let us look at some algorithms used in Decision Trees:

ID3 → (extension of D3)

C4.5 → (successor of ID3)

CART → (Classification And Regression Tree)

CHAID → (Chi-square automatic interaction detection Performs multi-level splits when computing classification trees)

MARS → (multivariate adaptive regression splines)

The ID3 algorithm builds decision trees using a top-down [greedy search](#) approach through the space of possible branches with no backtracking. A greedy algorithm, as the name suggests, always makes the choice that seems to be the best at that moment.

Steps in ID3 algorithm:

1. It begins with the original set S as the root node.
2. On each iteration of the algorithm, it iterates through the very unused attribute of the set S and calculates **Entropy (H)** and **Information gain (IG)** of this attribute.
3. It then selects the attribute which has the smallest Entropy or Largest Information gain.
4. The set S is then split by the selected attribute to produce a subset of the data.
5. The algorithm continues to recur on each subset, considering only attributes never selected before.

Attribute Selection Measures

If the dataset consists of N attributes then deciding which attribute to place at the root or at different levels of the tree as internal nodes is a complicated step. By just randomly selecting any node to be the root can't solve the issue. If we follow a random approach, it may give us bad results with low accuracy.

For solving this attribute selection problem, researchers worked and devised some solutions. They suggested using some criteria like - **Entropy, Information gain, Gini index, Gain Ratio, Reduction in Variance**

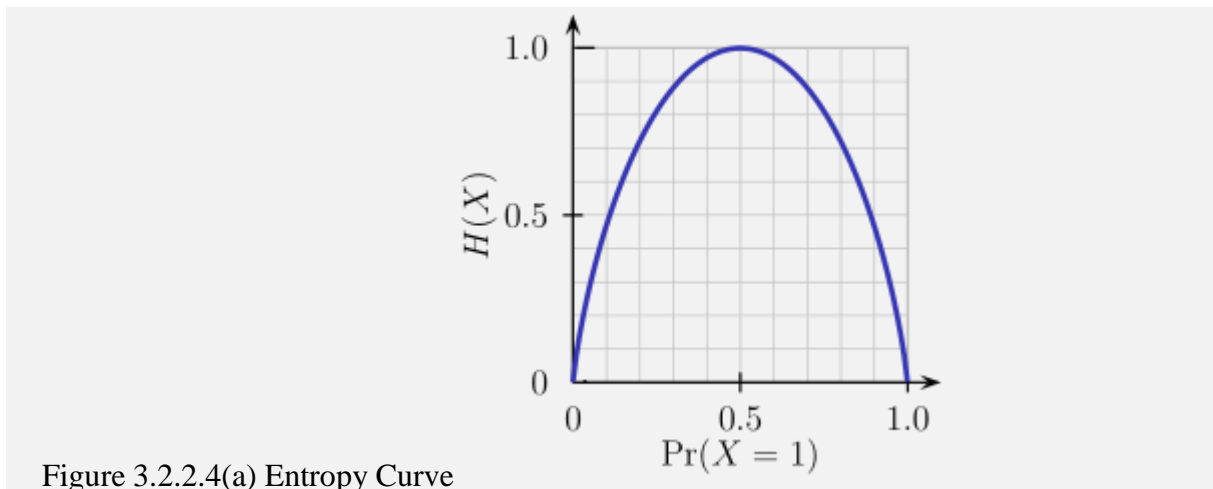
Chi-Square

These criterions will calculate values for every attribute. The values are sorted, and attributes are placed in the tree by following the order i.e, the attribute with a high value(in case of information gain) is placed at the root.

While using Information Gain as a criterion, we assume attributes to be categorical, and for the Gini index, attributes are assumed to be continuous.

3.2.2.4 Entropy

Entropy is a measure of the randomness in the information being processed. The higher the entropy, the harder it is to draw any conclusions from that information. Flipping a coin is an example of an action that provides information that is random.



From the above graph, it is quite evident that the entropy $H(X)$ is zero when the probability is either 0 or 1. The Entropy is maximum when the probability is 0.5 because it projects perfect randomness in the data and there is no chance if perfectly determining the outcome.

ID3 follows the rule - A branch with an entropy of zero is a leaf node and a branch with entropy more than zero needs further splitting.

Mathematically Entropy for 1 attribute is represented as:

Where $S \rightarrow$ Current state F

$$E(S) = \sum_{i=1}^c -p_i \log_2 p_i$$

Play Golf	
Yes	No
9	5



$$\begin{aligned} \text{Entropy(PlayGolf)} &= \text{Entropy}(5,9) \\ &= \text{Entropy}(0.36, 0.64) \\ &= -(0.36 \log_2 0.36) - (0.64 \log_2 0.64) \\ &= 0.94 \end{aligned}$$

Figure 3.2.2.4(b)

Where $S \rightarrow$ Current state, and $P_i \rightarrow$ Probability of an event i of state S or Percentage of class i in a node of state S .

Mathematically Entropy for multiple attributes is represented as:

$$E(T, X) = \sum_{c \in X} P(c)E(c)$$

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14



$$\begin{aligned}
 E(\text{PlayGolf, Outlook}) &= P(\text{Sunny}) * E(3,2) + P(\text{Overcast}) * E(4,0) + P(\text{Rainy}) * E(2,3) \\
 &= (5/14) * 0.971 + (4/14) * 0.0 + (5/14) * 0.971 \\
 &= 0.693
 \end{aligned}$$

where $T \rightarrow$ Current state and $X \rightarrow$ Selected attribute

3.2.2.5 Information Gain

Information gain or **IG** is a statistical property that measures how well a given attribute separates the training examples according to their target classification. Constructing a decision tree is all about finding an attribute that returns the highest information gain and the smallest entropy.

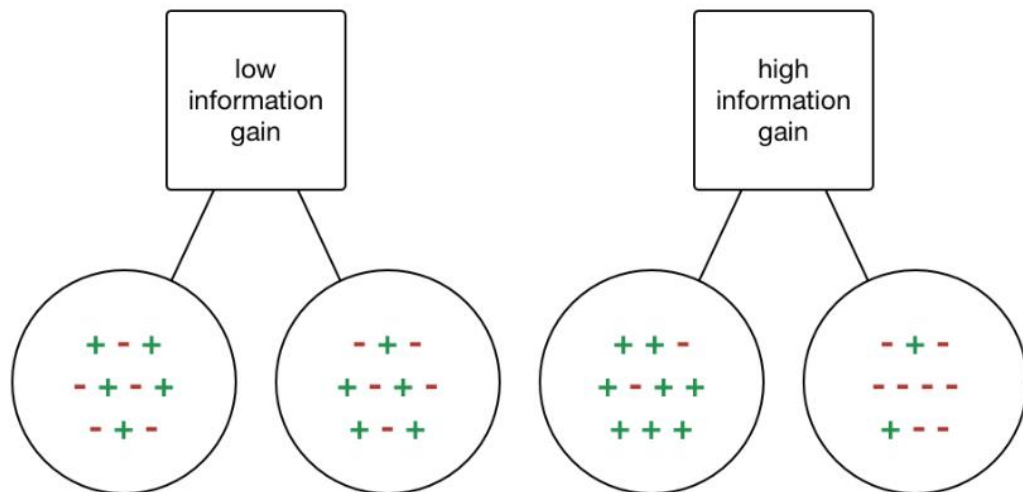


Figure 3.2.2.5(a).Information Gain

Information gain is a decrease in entropy. It computes the difference between entropy before split and average entropy after split of the dataset based on given attribute values. ID3 (Iterative Dichotomiser) decision tree algorithm uses information gain.

Mathematically, IG is represented as:

$$\text{Information Gain}(T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)$$

$$\begin{aligned} \text{IG}(\text{PlayGolf}, \text{Outlook}) &= E(\text{PlayGolf}) - E(\text{PlayGolf}, \text{Outlook}) \\ &= 0.940 - 0.693 \\ &= 0.247 \end{aligned}$$

In a much simpler way, we can conclude that:

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

Information Gain

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Gini Index

You can understand the Gini index as a cost function used to evaluate splits in the dataset. It is calculated by subtracting the sum of the squared probabilities of each class from one. It favors larger partitions and easy to implement whereas information gain favors smaller partitions with distinct values.

$$Gini = 1 - \sum_{i=1}^C (p_i)^2$$

Figure 3.2.2.5(b)

Gini Index works with the categorical target variable “Success” or “Failure”. It performs only Binary splits.

Higher the value of Gini index higher the homogeneity.

Steps to Calculate Gini index for a split

1. Calculate Gini for sub-nodes, using the above formula for success(p) and failure(q) (p^2+q^2).
2. Calculate the Gini index for split using the weighted Gini score of each node of that split.

CART (Classification and Regression Tree) uses the Gini index method to create split points.

Gain ratio

Information gain is biased towards choosing attributes with a large number of values as root nodes. It means it prefers the attribute with a large number of distinct values.

C4.5, an improvement of ID3, uses Gain ratio which is a modification of Information gain that reduces its bias and is usually the best option. Gain ratio overcomes the problem with information gain by taking into account the number of branches that would result before making the split. It corrects information gain by taking the intrinsic information of a split into account.

Let us consider if we have a dataset that has users and their movie genre preferences based on variables like gender, group of age, rating, etc. With the help of information gain, you split at ‘Gender’ (assuming it has the highest information gain) and now the variables ‘Group of Age’ and ‘Rating’ could be equally important and with the help of gain ratio, it will penalize a variable with more distinct values which will help us decide the split at the next level.

$$Gain\ Ratio = \frac{Information\ Gain}{SplitInfo} = \frac{Entropy\ (before) - \sum_{j=1}^K Entropy(j,\ after)}{\sum_{j=1}^K w_j \log_2 w_j}$$

Figure 3.2.2.5(c)

Where “before” is the dataset before the split, K is the number of subsets generated by the split, and (j, after) is subset j after the split.

Reduction in Variance

Reduction in variance is an algorithm used for continuous target variables (regression problems). This algorithm uses the standard formula of variance to choose the best split. The split with lower variance is selected as the criteria to split the population:

$$\text{Variance} = \frac{\sum (X - \bar{X})^2}{n}$$

Above X-bar is the mean of the values, X is actual and n is the number of values.

Steps to calculate Variance:

1. Calculate variance for each node.
2. Calculate variance for each split as the weighted average of each node variance.

Chi-Square

The acronym CHAID stands for *Chi*-squared Automatic Interaction Detector. It is one of the oldest tree classification methods. It finds out the statistical significance between the differences between sub-nodes and parent node. We measure it by the sum of squares of standardized differences between observed and expected frequencies of the target variable.

It works with the categorical target variable “Success” or “Failure”. It can perform two or more splits. Higher the value of Chi-Square higher the statistical significance of differences between sub-node and Parent node.

It generates a tree called CHAID (Chi-square Automatic Interaction Detector).

Mathematically, Chi-squared is represented as:

$$\chi^2 = \sum \frac{(O - E)^2}{E}$$

Where:

χ^2 = Chi Square obtained
 \sum = the sum of
 O = observed score
 E = expected score

Steps to Calculate Chi-square for a split:

1. Calculate Chi-square for an individual node by calculating the deviation for Success and Failure both
2. Calculated Chi-square of Split using Sum of all Chi-square of success and Failure of each node of the split

Avoiding Overfitting in Decision Trees

The common problem with Decision trees, especially having a table full of columns, they fit a lot. Sometimes it looks like the tree memorized the training data set. If there is no limit set on a decision tree, it will give you 100% accuracy on the training data set because in the worse case it will end up making 1 leaf for each observation. Thus this affects the accuracy when predicting samples that are not part of the training set.

Here are two ways to remove overfitting:

1. Pruning Decision Trees.
2. Random Forest

Pruning Decision Trees

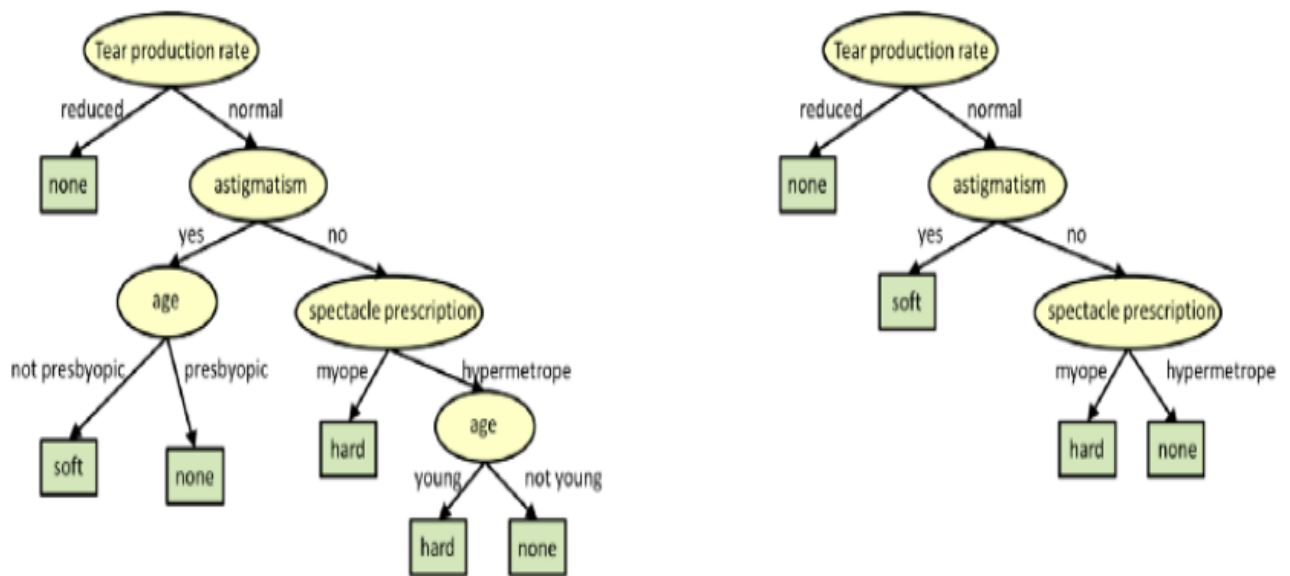
The splitting process results in fully grown trees until the stopping criteria are reached.

But, the fully grown tree is likely to overfit the data, leading to poor accuracy on unseen data.



Figure 3.2.2.5(d).Pruning in action

In pruning, you trim off the branches of the tree, i.e., remove the decision nodes starting from the leaf node such that the overall accuracy is not disturbed. This is done by segregating the actual training set into two sets: training data set, D and validation data set, V . Prepare the decision tree using the segregated training data set, D . Then continue trimming the tree accordingly to optimize the accuracy of the validation data set, V .



Original Tree

Pruned Tree

Pruning

In the above diagram, the 'Age' attribute in the left-hand side of the tree has been pruned as it has more importance on the right-hand side of the tree, hence removing overfitting.

Random Forest

Random Forest is an example of ensemble learning, in which we combine multiple machine learning algorithms to obtain better predictive performance.

Two key concepts that give it the name random:

1. A random sampling of training data set when building trees.
2. Random subsets of features considered when splitting nodes.

A technique known as bagging is used to create an ensemble of trees where multiple training sets are generated with replacement.

In the bagging technique, a data set is divided into N samples using randomized sampling. Then, using a single learning algorithm a model is built on all samples. Later, the resultant predictions are combined using voting or averaging in parallel.

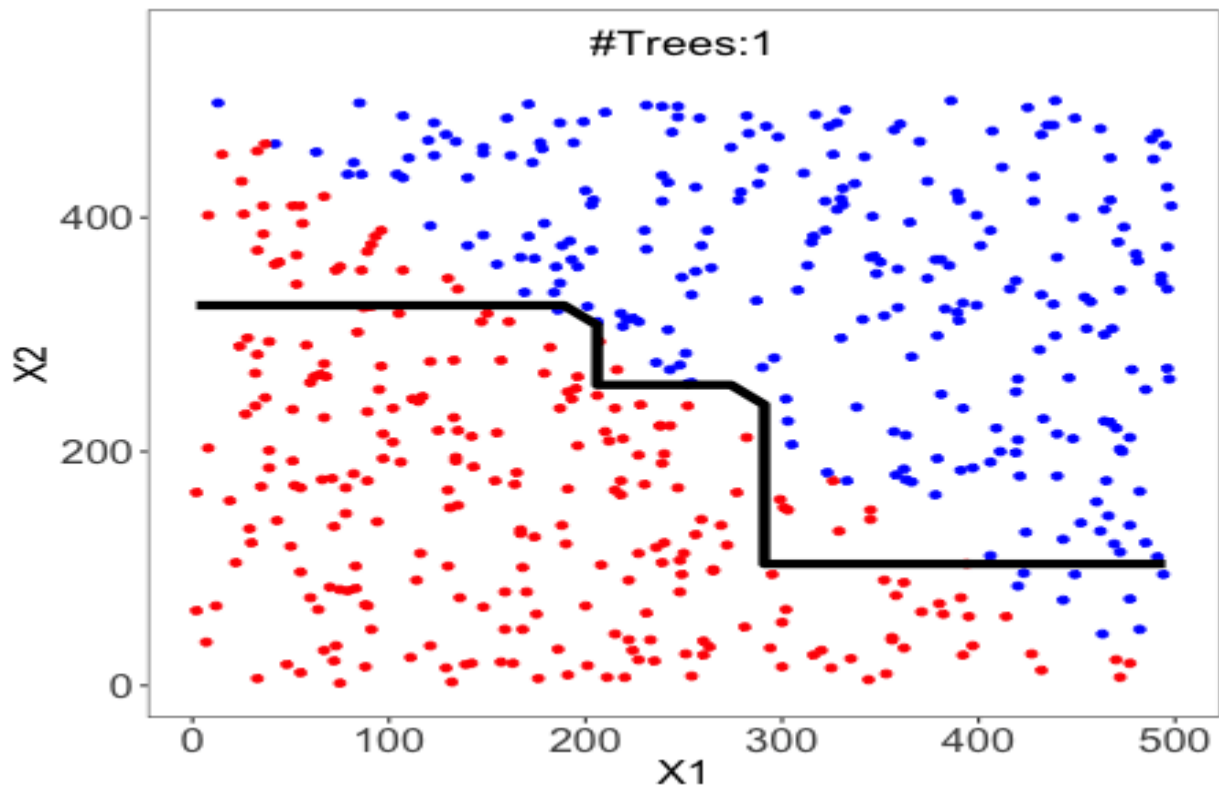


Figure 3.2.2.5(e).Random Forest in action

Decision Tree Classifier Building in Scikit-learn.

Load all the basic libraries.

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

Dataset

ssified as We will take only Age and EstimatedSalary as our independent variables X because of other features like Gender and User ID are irrelevant and have no effect on the purchasing capacity of a person. Purchased is our dependent variable y .

```
feature_cols = ['Age','EstimatedSalary' ]X = data.iloc[:,[2,3]].values
y = data.iloc[:,4].values
```

The next step is to split the dataset into training and test.

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size = 0.25, random_state= 0)
```

Perform feature scaling

```
#feature scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

Fit the model in the Decision Tree classifier.

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier()
classifier = classifier.fit(X_train,y_train)
```

Make predictions and check accuracy.

```
#prediction
y_pred = classifier.predict(X_test)#Accuracy
from sklearn import metricsprint('Accuracy Score:', metrics.accuracy_score(y_test,y_pred))
```

The decision tree classifier gave an accuracy of 91%.

Confusion Matrix

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)Output:
array([[64, 4],
       [ 2, 30]])
```

It means 6 observations have been cla false.

Let us first visualize the model prediction results.

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:,0].min()-1, stop= X_set[:,0].max()+1, step =
0.01),np.arange(start = X_set[:,1].min()-1, stop= X_set[:,1].max()+1, step = 0.01))
plt.contourf(X1,X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),
alpha=0.75, cmap = ListedColormap(("red","green")))plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())for i,j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set==j,0],X_set[y_set==j,1], c = ListedColormap(("red","green"))(i),label =
j)
plt.title("Decision Tree(Test set)")
plt.xlabel("Age")
plt.ylabel("Estimated Salary")
```



Let us also visualize the tree:

You can use Scikit-learn's `export_graphviz` function to display the tree within a Jupyter notebook. For plotting trees, you also need to install Graphviz and pydotplus.

```
conda install python-graphviz
```

```
pip install pydotplus
```

`export_graphviz` function converts decision tree classifier into dot file and pydotplus convert this dot file to png or displayable form on Jupyter.

```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols, class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```


In the decision tree chart, each internal node has a decision rule that splits the data. Gini referred to as the Gini ratio, which measures the impurity of the node. You can say a node is pure when all of its records belong to the same class, such nodes known as the leaf node.

Here, the resultant tree is unpruned. This unpruned tree is unexplainable and not easy to understand. In the next section, let's optimize it by pruning.

3.2.2.6 Optimizing the Decision Tree Classifier

criterion: optional (default="gini") or Choose attribute selection measure: This parameter allows us to use the different-different attribute selection measure. Supported criteria are "gini" for the Gini index and "entropy" for the information gain.

splitter: string, optional (default="best") or Split Strategy: This parameter allows us to choose the split strategy. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth: int or None, optional (default=None) or Maximum Depth of a Tree: The maximum depth of the tree. If None, then nodes are expanded until all the leaves contain less than min_samples_split samples. The higher value of maximum depth causes overfitting, and a lower value causes underfitting (Source).

In Scikit-learn, optimization of decision tree classifier performed by only pre-pruning. The maximum depth of the tree can be used as a control variable for pre-pruning.

```
# Create Decision Tree classifier object
classifier = DecisionTreeClassifier(criterion="entropy", max_depth=3) # Train Decision Tree Classifier
classifier = classifier.fit(X_train, y_train) # Predict the response for test dataset
```

```
y_pred = classifier.predict(X_test)# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Well, the classification rate increased to 94%, which is better accuracy than the previous model.

Now let us again visualize the pruned Decision tree after optimization.

```
dot_data = StringIO()
export_graphviz(classifier, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True, feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

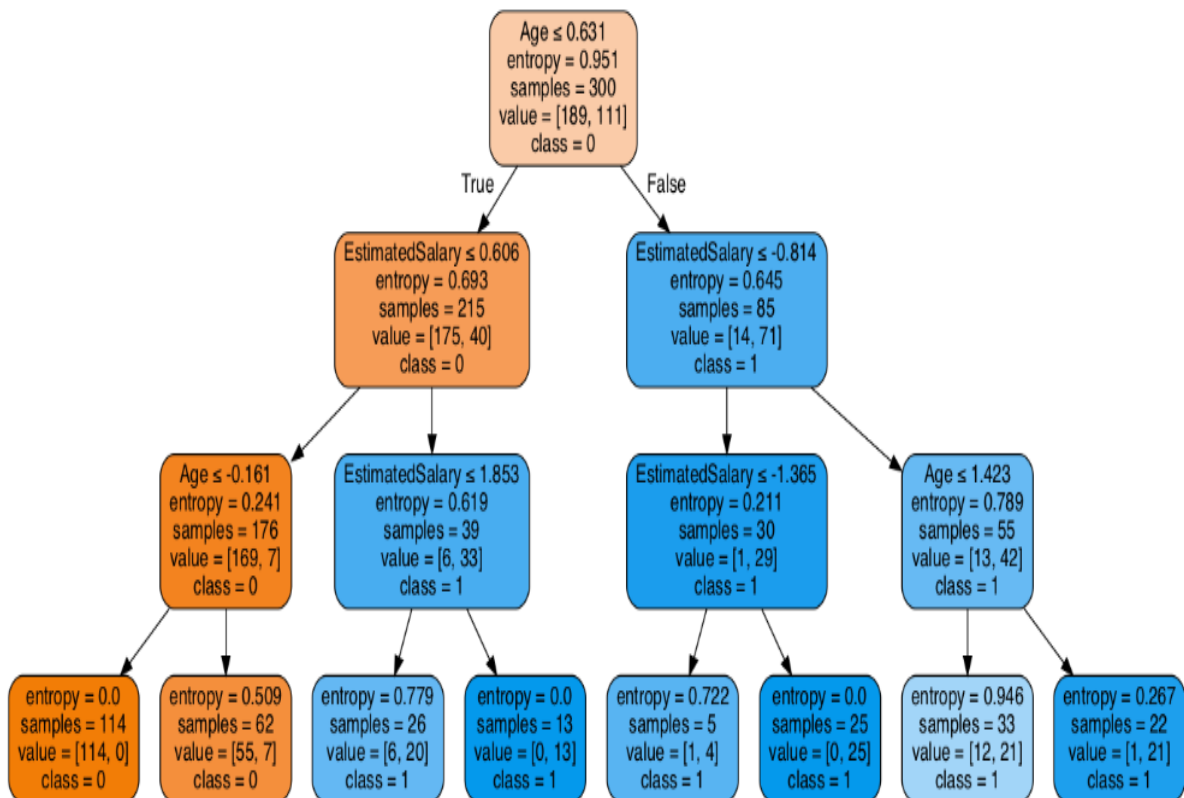


Figure 3.2.2.5(f).Decision Tree after pruning

This pruned model is less complex, explainable, and easy to understand than the previous decision tree model plot.

3.2.2.7 Decision Tree using numpy and panda

```

def DecisionTree():
    from sklearn import tree
    clf3 = tree.DecisionTreeClassifier() # empty model of the decision tree
    clf3 = clf3.fit(X,y)
    # calculating accuracy-----
    from sklearn.metrics import accuracy_score
    y_pred=clf3.predict(X_test)
    print(accuracy_score(y_test, y_pred))
    print(accuracy_score(y_test, y_pred,normalize=False))

    psymptoms=[Symptom1.get(),Symptom2.get(),Symptom3.get(),Symptom4.get(),Sym
ptom5.get
    ()]
    for k in range(0,len(l1)):
        for z in psymptoms:
            if(z==l1[k]):
                l2[k]=1

    inputtest=[l2]
    predict = clf3.predict(inputtest)
    predicted=predict[0]
    h='no'
    for a in range(0,len(disease)):
        if(predicted == a):
            h='yes'
            break
    if(h=='yes'):
        t1.delete("1.0", END)
        t1.insert(END, disease[a])
    else:
        t1.delete("1.0", END)
        t1.insert(END, "Not Found")

```

3.2.3 Random Forest

Random forest is a flexible, easy to use machine learning algorithm that produces, even without hyper-parameter tuning, a great result most of the time. It is also one of the most used algorithms, because of its simplicity and diversity (it can be used for both classification and regression tasks). In this post we'll learn how the random forest algorithm works, how it differs from other algorithms and how to use it.

A big part of machine learning is classification — we want to know what class (a.k.a. group) an observation belongs to. The ability to precisely classify observations is extremely valuable for various business applications like predicting whether a particular user will buy a product or forecasting whether a given loan will default or not.

Data science provides a plethora of classification algorithms such as logistic regression, support vector machine, naive Bayes classifier, and decision trees. But near the top of the classifier hierarchy is the random forest classifier (there is also the random forest regressor but that is a topic for another day).

In this we will examine how basic decision trees work, how individual decisions trees are combined to make a random forest, and ultimately discover why random forests are so good at what they do.

3.2.3.1 Decision Trees

Let's quickly go over decision trees as they are the building blocks of the random forest model. Fortunately, they are pretty intuitive. I'd be willing to bet that most people have used a decision tree, knowingly or not, at some point in their lives.

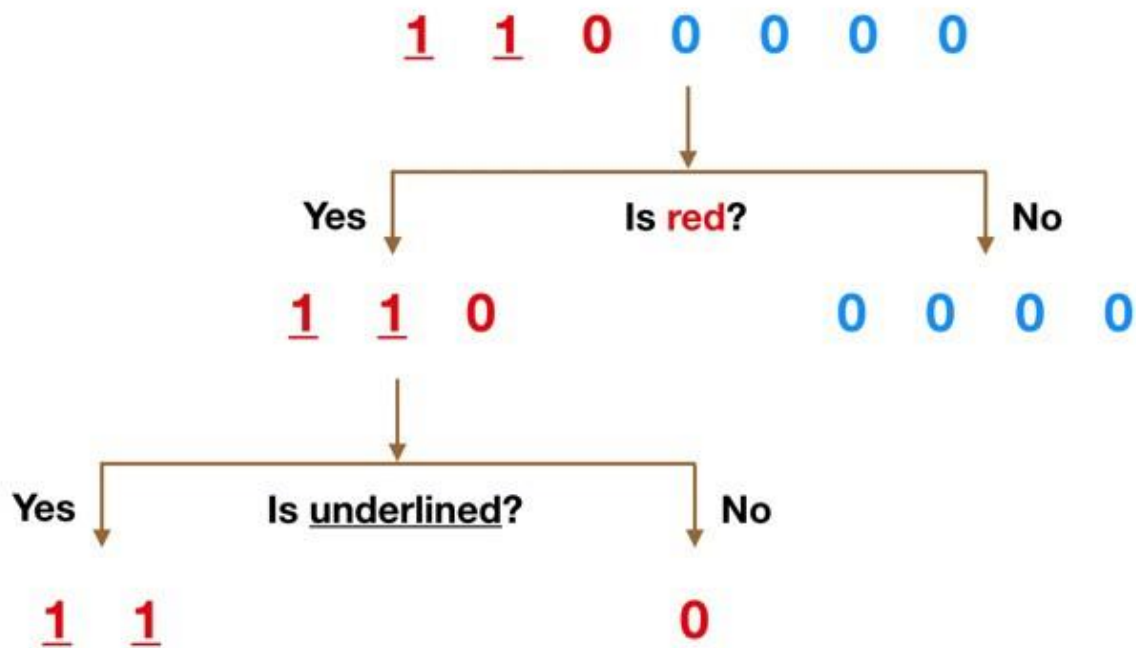


Figure 3.2.3.1(a)

It's probably much easier to understand how a decision tree works through an example.

Imagine that our dataset consists of the numbers at the top of the figure to the left. We have two 1s and five 0s (1s and 0s are our classes) and desire to separate the classes using their features. The features are color (red vs. blue) and whether the observation is underlined or not. So how can we do this?

Color seems like a pretty obvious feature to split by as all but one of the 0s are blue. So we can use the question, “Is it red?” to split our first node. You can think of a node in a tree as the point where the path splits into two — observations that meet the criteria go down the Yes branch and ones that don't go down the No branch.

The No branch (the blues) is all 0s now so we are done there, but our Yes branch can still be split further. Now we can use the second feature and ask, “Is it underlined?” to make a second split.

The two 1s that are underlined go down the Yes subbranch and the 0 that is not underlined goes down the right subbranch and we are all done. Our decision tree was able to use the two features to split up the data perfectly.

Obviously in real life our data will not be this clean but the logic that a decision tree employs remains the same.

3.2.3.2 The Random Forest Classifier

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction (see figure below).

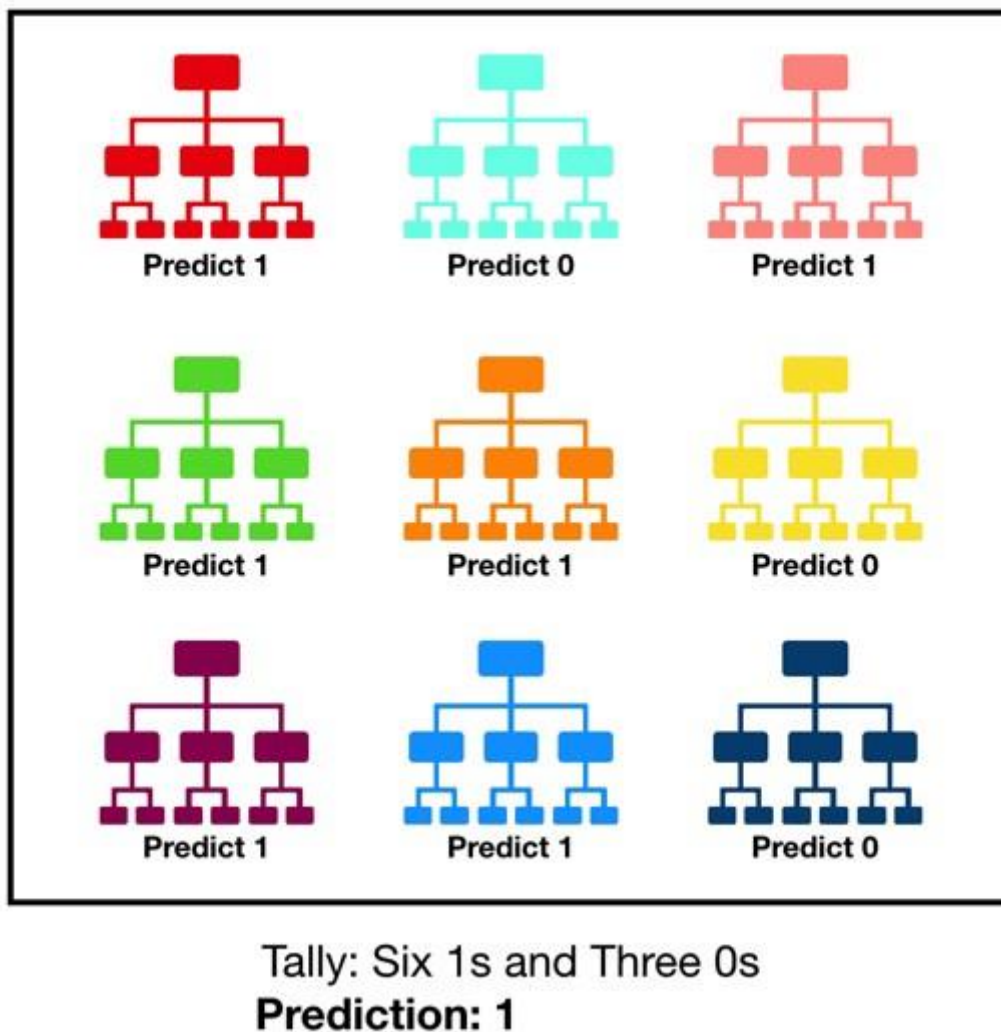


Figure 3.2.3.2(b)

The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds. In data science speak, the reason that the random forest model works so well is:

A large number of relatively uncorrelated models (trees) operating as a committee will outperform any of the individual constituent models.

The low correlation between models is the key. Just like how investments with low correlations (like stocks and bonds) come together to form a portfolio that is greater than the sum of its parts, uncorrelated models can produce ensemble predictions that are more accurate than any of the individual predictions. The reason for this wonderful effect is that the trees protect each other from their individual errors (as long as they don't constantly all err in the same direction). While some trees may be wrong, many other trees will be right, so as a group the trees are able to move in the correct direction. So the prerequisites for random forest to perform well are:

1. There needs to be some actual signal in our features so that models built using those features do better than random guessing.
2. The predictions (and therefore the errors) made by the individual trees need to have low correlations with each other.

3.2.2.3 An Implementation and Explanation of the Random Forest in Python

def randomforest():

from sklearn.ensemble import RandomForestClassifier

clf4 = RandomForestClassifier()

clf4 = clf4.fit(X,np.ravel(y))

calculating accuracy-----

from sklearn.metrics import accuracy_score

y_pred=clf4.predict(X_test)

print(accuracy_score(y_test, y_pred))

print(accuracy_score(y_test, y_pred,normalize=False))

psymptoms =

[Symptom1.get(),Symptom2.get(),Symptom3.get(),Symptom4.get(),Symptom5.get()]

```

for k in range(0,len(l1)):

    for z in psymptoms:

        if(z==l1[k]):

            l2[k]=1


inputtest = [l2]

predict = clf4.predict(inputtest)

predicted=predict[0]


h='no'

for a in range(0,len(disease)):

    if(predicted == a):

        h='yes'

        break


if (h=='yes'):

    t2.delete("1.0", END)

    t2.insert(END, disease[a])

else:

    t2.delete("1.0", END)

    t2.insert(END, "Not Found")

```

3.2.2.4 Random Forest

The random forest is a model made up of many decision trees. Rather than just simply averaging the prediction of trees (which we could call a “forest”), this model uses two key concepts that gives it the name random:

1. Random sampling of training data points when building trees
2. Random subsets of features considered when splitting nodes

Random sampling of training observations

When training, each tree in a random forest learns from a random sample of the data points. The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree. The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training data, overall, the entire forest will have lower variance but not at the cost of increasing the bias.

At test time, predictions are made by averaging the predictions of each decision tree. This procedure of training each individual learner on different bootstrapped subsets of the data and then averaging the predictions is known as bagging, short for bootstrap aggregating.

3.2.2.5 Random Forest in Practice

Next, we’ll build a random forest in Python using Scikit-Learn. Instead of learning a simple problem, we’ll use a real-world dataset split into a training and testing set. We use a test set as an estimate of how the model will perform on new data which also lets us determine how much the model is overfitting.

Dataset

The problem we’ll solve is a binary classification task with the goal of predicting an individual’s health. The features are socioeconomic and lifestyle characteristics of individuals and the label is 0 for poor health and 1 for good health. This dataset was collected by the

Centers for Disease Control and Prevention and is available [here](#).

Table 3.2.2.5(a)

	_STATE	FMONTH	IDATE	IMONTH	IDAY	IYEAR	DISPCODE	SEQNO	_PSU	CTELENUM	...
383119	49.0	4.0	b'05192015'	b'05'	b'19'	b'2015'	1100.0	2.015009e+09	2.015009e+09	NaN	...
55536	9.0	9.0	b'09232015'	b'09'	b'23'	b'2015'	1100.0	2.015005e+09	2.015005e+09	1.0	...
267093	34.0	10.0	b'11052015'	b'11'	b'05'	b'2015'	1100.0	2.015011e+09	2.015011e+09	NaN	...
319092	41.0	4.0	b'04062015'	b'04'	b'06'	b'2015'	1100.0	2.015002e+09	2.015002e+09	1.0	...
420978	54.0	5.0	b'05112015'	b'05'	b'11'	b'2015'	1100.0	2.015004e+09	2.015004e+09	NaN	...

Generally, 80% of a data science project is spent cleaning, exploring, and making features out of the data. However, for this article, we'll stick to the modeling. (For details of the other steps).

This is an imbalanced classification problem, so accuracy is not an appropriate metric. Instead we'll measure the Receiver Operating Characteristic Area Under the Curve (ROC AUC), a measure from 0 (worst) to 1 (best) with a random guess scoring 0.5. We can also plot the ROC curve to assess a model.

The notebook contains the implementation for both the decision tree and the random forest, but here we'll just focus on the random forest. After reading in the data, we can instantiate and train a random forest as follows:

After a few minutes to train, the model is ready to make predictions on the testing data.

3.3 Display

The display after applying the above widgets show various different actions. The actions are described in the below explanation. For an web application, one should allows be user-friendly. That means it should always be easy to use and direct application with high through-put or output which directly means easy UI/UX. The simple UI/UX is portrayed As, UI design (UI) or user interface engineering is the design of user interfaces for machines and software, such as computers, home appliances, mobile gadgets, and other electronic gadgets, with the attention on maximizing usability and the user experience. The objective of UI configuration is to make the client's collaboration as basic and effective as would be prudent, as far as achieving client objectives (client focused plan).

Great UI configuration encourages completing the job that needs to be done without attracting superfluous regard for itself. Graphic design and typography are used to help

its usability, affecting how the client plays out specific cooperations and improving the tasteful intrigue of the structure; plan feel may upgrade or degrade the capacity of clients to utilize the elements of the interface. The plan procedure must adjust specialized usefulness and visual components (e.g., mental model) to make a framework that isn't just operational yet additionally usable and versatile to changing client needs.

Interface configuration is associated with a wide scope of ventures from PC frameworks, to autos, to business planes; these activities include a significant part of a similar essential human collaborations yet likewise require some novel abilities and learning. Accordingly, fashioners will in general have practical experience in specific sorts of activities and have aptitudes focused on their ability, regardless of whether that be software plan, client research, web structure, or industrial plan.

The screenshot shows a web application titled "Disease Predictor using Machine Learning". The interface includes a form for patient information and symptom selection. At the top, there is a green header bar with the title. Below the header, there is a label "Name of the Patient" followed by a yellow input field. Underneath, there are five labels "Symptom 1" through "Symptom 5", each followed by a dropdown menu currently set to "None". Below the symptom section, there are three red buttons labeled "DecisionTree", "RandomForest", and "NaiveBayes". To the right of these buttons are three yellow input fields. On the far right, there are four green buttons labeled "DecisionTree", "Randomforest", "NaiveBayes", and "Generate Report".

Figure 3.3(a)

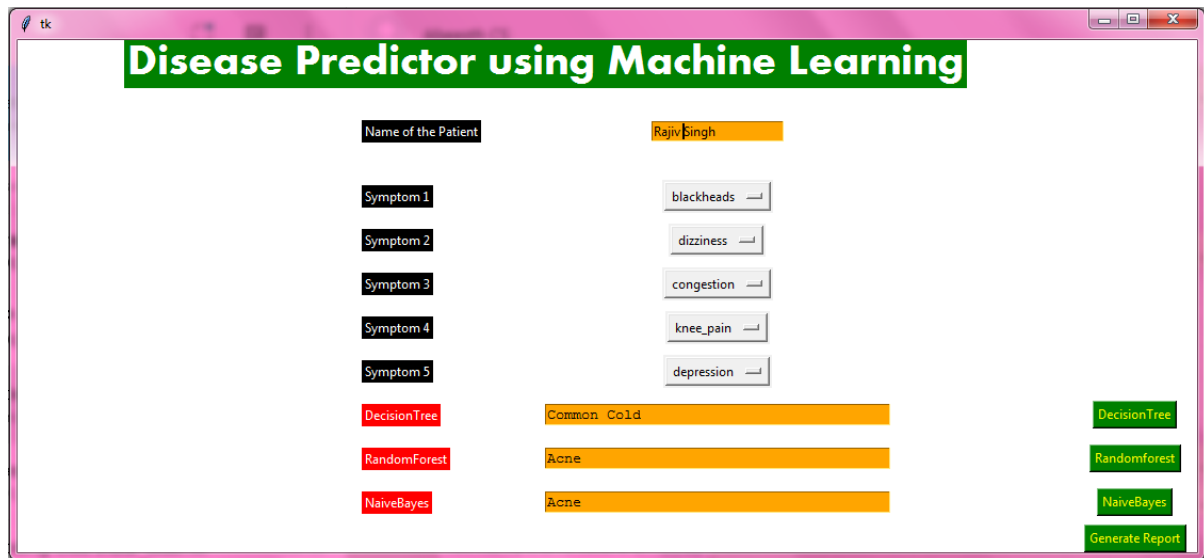


Figure 3.3(b)

gui_stuff-----

```
root = Tk()
```

```
root.configure(background='white')
```

```
# entry variables
```

```
Symptom1 = StringVar()
```

```
Symptom1.set(None)
```

```
Symptom2 = StringVar()
```

```
Symptom2.set(None)
```

```
Symptom3 = StringVar()
```

```
Symptom3.set(None)
```

```
Symptom4 = StringVar()
```

```
Symptom4.set(None)
```

```
Symptom5 = StringVar()
```

```
Symptom5.set(None)
```

```
Name = StringVar()
```

```
# Heading
```

```
w2 = Label(root, justify=CENTER, text="Disease Predictor using Machine Learning",  
fg="white", bg="green")
```

```
w2.config(font=("Aharoni", 30))
```

```
w2.grid(row=1, column=0, columnspan=2, padx=100)
```

```
#w2 = Label(root, justify=LEFT, text="A Project by Varun Agrawal", fg="white",  
bg="green")
```

```
#w2.config(font=("Aharoni", 30))
```

```
#w2.grid(row=2, column=0, columnspan=2, padx=100)
```

```
# labels
```

```
NameLb = Label(root, text="Name of the Patient", fg="white", bg="black")
```

```
NameLb.grid(row=6, column=1, pady=30, sticky=W)
```

```
S1Lb = Label(root, text="Symptom 1", fg="white", bg="black")
```

```
S1Lb.grid(row=7, column=1, pady=10, sticky=W)
```

```
S2Lb = Label(root, text="Symptom 2", fg="white", bg="black")
```

```
S2Lb.grid(row=8, column=1, pady=10, sticky=W)
```

```
S3Lb = Label(root, text="Symptom 3", fg="white", bg="black")
```

```
S3Lb.grid(row=9, column=1, pady=10, sticky=W)
```

```
S4Lb = Label(root, text="Symptom 4", fg="white", bg="black")
```

```
S4Lb.grid(row=10, column=1, pady=10, sticky=W)
```

```
S5Lb = Label(root, text="Symptom 5", fg="white", bg="black")
```

```
S5Lb.grid(row=11, column=1, pady=10, sticky=W)
```

```
lrLb = Label(root, text="DecisionTree", fg="white", bg="red")
```

```
lrLb.grid(row=15, column=1, pady=10, sticky=W)
```

```
destreeLb = Label(root, text="RandomForest", fg="white", bg="red")
```

```
destreeLb.grid(row=17, column=1, pady=10, sticky=W)
```

```
ranfLb = Label(root, text="NaiveBayes", fg="white", bg="red")
```

```
ranfLb.grid(row=19, column=1, pady=10, sticky=W)
```

```
# entries
```

```
OPTIONS = sorted(11)
```

```
NameEn = Entry(root, textvariable=Name, bg="orange", fg="black")
```

```
NameEn.grid(row=6, column=1)
```

```
S1En = OptionMenu(root, Symptom1, *OPTIONS)
```

```
S1En.grid(row=7, column=1)
```

```
S2En = OptionMenu(root, Symptom2,*OPTIONS)
```

```
S2En.grid(row=8, column=1)
```

```
S3En = OptionMenu(root, Symptom3,*OPTIONS)
```

```
S3En.grid(row=9, column=1)
```

```
S4En = OptionMenu(root, Symptom4,*OPTIONS)
```

```
S4En.grid(row=10, column=1)
```

```
S5En = OptionMenu(root, Symptom5,*OPTIONS)
```

```
S5En.grid(row=11, column=1)
```

```
genReport = Button(root, text="Generate Report",  
command=generateReport,bg="green",fg="yellow")
```

```
genReport.grid(row=21, column=2,padx=10)
```

```
dst = Button(root, text="DecisionTree", command=DecisionTree,bg="green",fg="yellow")
```

```
dst.grid(row=15, column=2,padx=10)
```

```
rnf = Button(root, text="Randomforest", command=randomforest,bg="green",fg="yellow")
```

```
rnf.grid(row=17, column=2,padx=10)
```

```
lr = Button(root, text="NaiveBayes", command=NaiveBayes,bg="green",fg="yellow")
```

```
lr.grid(row=19, column=2,padx=10)
```

```
#textfileds
```

```
t1 = Text(root, height=1, width=40,bg="orange",fg="black")
```

```
t1.grid(row=15, column=1, padx=10)
```

```
t2 = Text(root, height=1, width=40,bg="orange",fg="black")
```

```
t2.grid(row=17, column=1 , padx=10)
```

```
t3 = Text(root, height=1, width=40,bg="orange",fg="black")
```

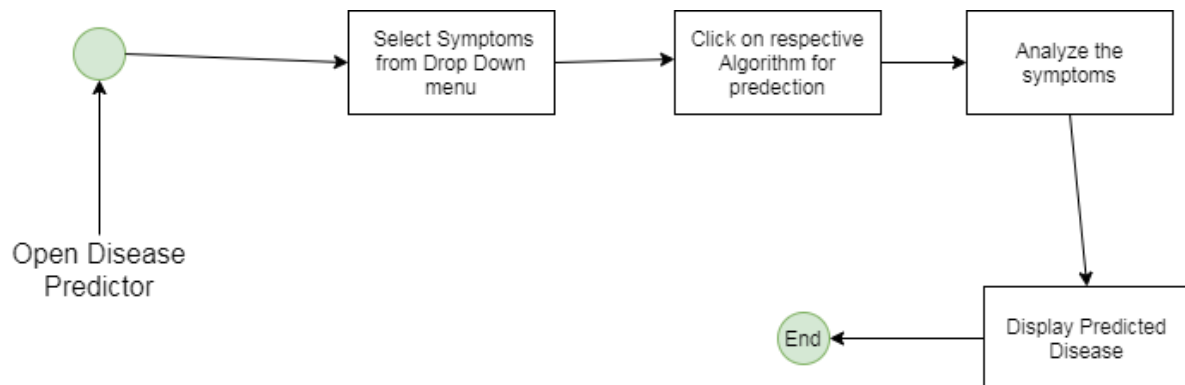
```
t3.grid(row=19, column=1 , padx=10)
```

```
root.mainloop()
```

3.4 State Diagram

It explains different state of the system. First the user opens Disease Predictor. The user selects the symptoms. When finished selecting symptoms the user submits the symptoms. Disease Predictor analyzes the symptoms and displays the result

Figure 3.4(a)



3.5 Sequence diagram

It explains the sequence of the Disease Predictor. Initially system shows the symptoms to be selected. The user selects the symptoms and submits to the system. The Disease Predictor predicts and displays the result.

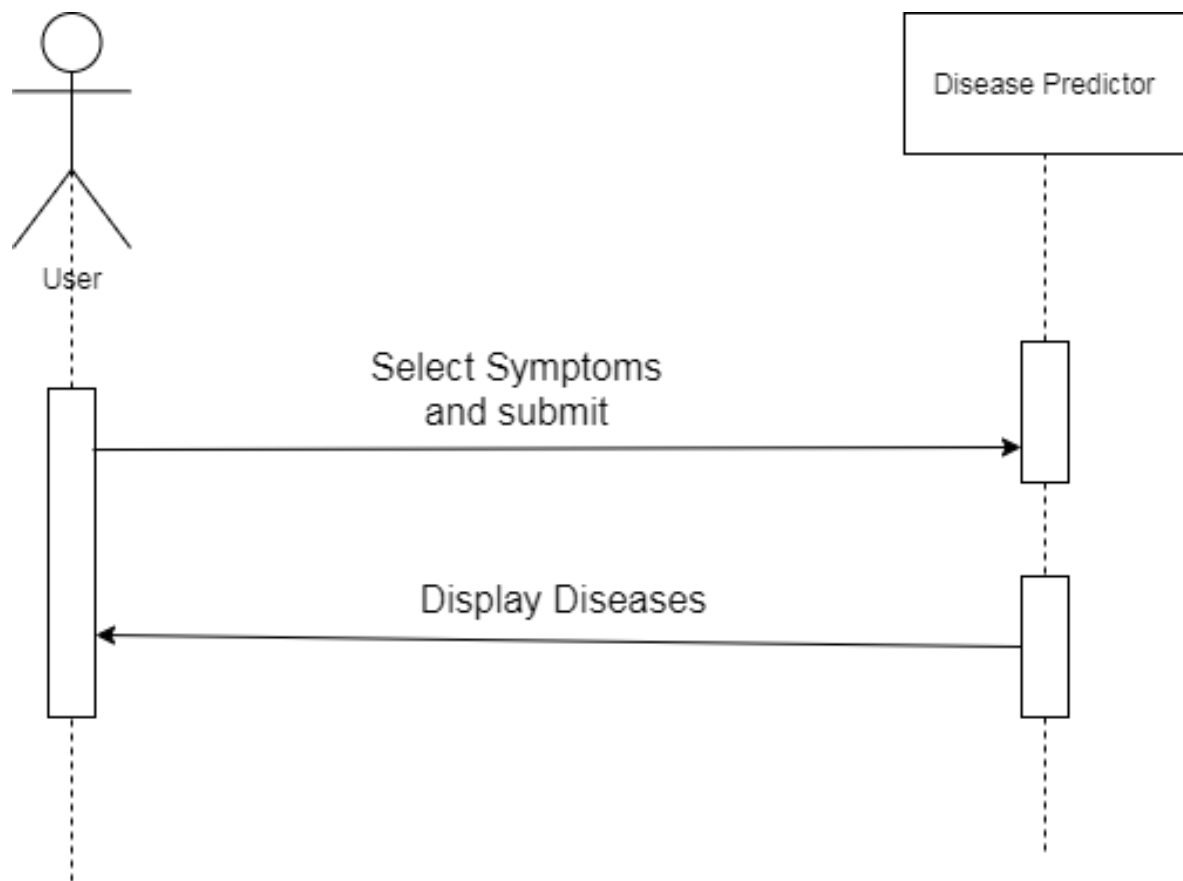


Figure 3.5(a)

Chapter 4

Testing & results

Unit testing is an incredible order which can prompt 40%-80% decreases underway bug thickness. Unit testing additionally has a few other significant advantages: Improves your application design and viability.

Prompts better APIs and composability by concentrating designers on the engineer involvement (API) before execution subtleties.

Gives snappy criticism on record spare to disclose to you whether your progressions worked. This can supplant `console.log()` and clicking around in the UI to test changes. Newcomers to unit testing may spend an additional 15% - 30% on the TDD procedure as they make sense of how to test different segments, however experienced TDD experts may encounter funds in usage time utilizing TDD.

Gives an extraordinary security net which can upgrade your certainty when its opportunity to include highlights or refactor existing highlights. In any case, a few things are simpler to unit test than others. In particular, unit tests work extraordinary for unadulterated capacities: Functions which given a similar information, dependably return a similar yield, and have no symptoms.

Frequently, UI segments don't fall into that classification of things which are anything but difficult to unit test, which makes it harder to adhere to the control of TDD: Writing tests first.

Composing tests initially is important to accomplish a portion of the advantages I recorded: engineering enhancements, better designer experience plan, and speedier criticism as you're building up your application. It takes control and practice to prepare yourself to utilize TDD.

Numerous engineers like to tinker before they compose tests, yet on the off chance that you don't compose tests first, you deny yourself of a great deal of the best highlights of unit tests.

It merits the training and control, however. TDD with unit tests can prepare you to compose UI segments which are far more straightforward, simpler to keep up, and simpler to form and reuse with different parts.

One ongoing advancement in my testing discipline is the improvement of the RITEway unit testing structure, which is a little wrapper around Tape that causes you compose less complex, increasingly viable tests.

Regardless of what structure you use, the accompanying tips will enable you to compose better, progressively testable, increasingly clear, increasingly composable UI segments:

Support unadulterated parts for UI code: given same props, dependably render a similar segment. On the off chance that you need state from the application, you can wrap those unadulterated parts with a compartment segment which oversees state and symptoms.

Seclude application rationale/business leads in unadulterated reducer capacities.

Detach reactions utilizing compartment segments.

Support Pure Components

An unadulterated part is a segment which, given similar props, dependably renders the equivalent UI, and has no reactions. E.g.,

```
import React from 'respond';

const Hello = ({ userName }) => (

<div className="greeting">Hello, {userName}!</div>

);
```

send out default Hello;

These sorts of parts are commonly exceptionally simple to test. You'll require an approach to choose the segment (for this situation, we're choosing by the welcome className), and you'll have to know the normal yield. To compose unadulterated segment tests, I use render-part from RITEway.

To begin, introduce RITEway:

```
npm introduce - spare dev riteway
```

Inside, RITEway utilizes respond dom/server renderToStaticMarkup() and envelops the yield by a Cheerio object for simple choices. In case you're not utilizing RITEway, you can do all that physically to make your very own capacity to render React segments to static markup you can inquiry with Cheerio.

When you have a render capacity to deliver a Cheerio object from your markup, you can compose part tests this way:

```
import { depict } from 'riteway';

import render from 'riteway/render-segment';

import React from 'respond';

import Hello from './hi';

describe('Hello segment', async attest => {

  const userName = 'Spiderman';

  const $ = render(<Hello userName={userName}/>);
```

```

assert({

given: 'a username',

should: 'Render a welcome to the right username.',

genuine: $('greeting')

.html()

.trim(),

anticipated: 'Hi, ${userName}!'

});

});

```

Be that as it may, that is not extremely intriguing. Imagine a scenario in which you have to test a stateful part, or a segment with reactions. That is the place TDD gets truly fascinating for React parts, in light of the fact that the response to that question is equivalent to the response to another significant inquiry: "How might I make my React segments progressively viable and simple to investigate?"

The appropriate response: Isolate your state and reactions from your introduction parts. You can do that by exemplifying your state and reaction the board in a compartment segment, and afterward pass the state into an unadulterated segment through props.

In any case, didn't the snares API cause it with the goal that we to can have level segment chains of command and disregard all that segment settling stuff? Indeed, not exactly. It's as yet a smart thought to keep your code in three distinct cans, and keep these pails segregated from one another:

Show/UI Components

Program rationale/business rules—the stuff that manages the issue you're understanding for the client.

Reactions (I/O, organize, circle, and so forth.)

I would say, on the off chance that you keep the showcase/UI concerns separate from program rationale and symptoms, it makes your life significantly simpler. This standard guideline has constantly remained constant for me, in each language and each structure I've at any point utilized, incorporating React with snares.

How about we show stateful segments by structure a tick counter. To start with, we'll construct the UI segment. It should show something like, "Snaps: 13" to disclose to you how often a catch has been clicked. The catch will simply say "Snap".

Unit tests for the presentation part are really simple. We actually just need to test that the catch gets rendered by any stretch of the imagination (we couldn't care less about what the name says—it may state various things in various dialects, contingent upon client region settings). We would like to ensure that the right number of snaps gets showed. We should compose two tests: One for the catch show, and one for the quantity of snaps to be rendered accurately.

When utilizing TDD, I every now and again utilize two unique attestations to guarantee that I've composed the part with the goal that the best possible esteem is pulled from props. It's conceivable to compose a test with the goal that you could hard-code the incentive in the capacity. To make preparations for that, you can compose two tests which each test an alternate esteem.

For this situation, we'll make a segment called `<ClickCounter>`, and that segment will have a prop for the snap tally, called `clicks`. To utilize it, just render the segment and set the `snaps` prop to the quantity of snaps you need it to show.

We should take a gander at a couple of unit tests that could guarantee we're pulling the snap check from props. We should make another document, `click-counter/click-counter-component.test.js`:

```
root = Tk()
root.configure(background='white')
```

```
# entry variables
Symptom1 = StringVar()
Symptom1.set(None)
Symptom2 = StringVar()
Symptom2.set(None)
Symptom3 = StringVar()
Symptom3.set(None)
Symptom4 = StringVar()
Symptom4.set(None)
Symptom5 = StringVar()
Symptom5.set(None)
Name = StringVar()
```

I like to make little manufacturing plant capacities to make it simpler to compose tests. For this situation, createCounter will take various snaps to infuse, and return a rendered part utilizing that number of snaps:

```
# Heading
w2 = Label(root, justify=CENTER, text="Disease Predictor using Machine Learning",
fg="white", bg="green")
w2.config(font=("Aharoni", 30))
w2.grid(row=1, column=0, columnspan=2, padx=100)
#w2 = Label(root, justify=LEFT, text="A Project by Varun Agrawal", fg="white", bg="green")
#w2.config(font=("Aharoni", 30))
#w2.grid(row=2, column=0, columnspan=2, padx=100)

# labels
NameLb = Label(root, text="Name of the Patient", fg="white", bg="black")
NameLb.grid(row=6, column=1, pady=30, sticky=W)
```

```
S1Lb = Label(root, text="Symptom 1", fg="white", bg="black")
```

```
S1Lb.grid(row=7, column=1, pady=10, sticky=W)
```

```
S2Lb = Label(root, text="Symptom 2", fg="white", bg="black")
```

```
S2Lb.grid(row=8, column=1, pady=10, sticky=W)
```

```
S3Lb = Label(root, text="Symptom 3", fg="white", bg="black")
```

```
S3Lb.grid(row=9, column=1, pady=10, sticky=W)
```

```
S4Lb = Label(root, text="Symptom 4", fg="white", bg="black")
```

```
S4Lb.grid(row=10, column=1, pady=10, sticky=W)
```

```
S5Lb = Label(root, text="Symptom 5", fg="white", bg="black")
```

```
S5Lb.grid(row=11, column=1, pady=10, sticky=W)
```

```
lrLb = Label(root, text="DecisionTree", fg="white", bg="red")
```

```
lrLb.grid(row=15, column=1, pady=10, sticky=W)
```

```
destreeLb = Label(root, text="RandomForest", fg="white", bg="red")
```

```
destreeLb.grid(row=17, column=1, pady=10, sticky=W)
```

```
ranfLb = Label(root, text="NaiveBayes", fg="white", bg="red")
```

```
ranfLb.grid(row=19, column=1, pady=10, sticky=W)
```

With the tests composed, it's a great opportunity to make our ClickCounter show segment. I've colocated mine in a similar organizer with my test document, with the name, click-counter-component.js. To start with, how about we compose a segment section and watch our test come up short:

```
import React, { Fragment } from 'react';
```



```
trade default () =>
```

```
<Fragment>
```

```
</Fragment>
```

```
;
```

On the off chance that we spare and run our tests, we'll get a `TypeError`, which right now triggers Node's `UnhandledPromiseRejectionWarning`—eventually, Node will stop with the aggravating admonitions with the additional section of `DeprecationWarning` and simply toss an `UnhandledPromiseRejectionError`. We get the `TypeError` in light of the fact that our determination returns invalid, and we're endeavoring to run `.trim()` on it.

```
import React, { Fragment } from 'respond';
```

```
send out default () =>
```

```
<Fragment>
```

```
<span className="clicks-count">3</span>
```

```
</Fragment>
```

```
;
```

Fantastic. Presently we ought to make them breeze through test, and one coming up short test:

```
# ClickCounter segment
```

```
alright 2 Given a tick tally: should render the right number of snaps.
```

```
not alright 3 Given a tick tally: should render the right number of snaps.
```

-

administrator: deepEqual

anticipated: 5

genuine: 3

at: attest (/home/eric/dev/respond unadulterated part
starter/node_modules/riteway/source/riteway.js:15:10)

...

To fix it, accept the consider a prop, and utilize the live prop an incentive in the JSX:

```
import React, { Fragment } from 'respond';
```

```
send out default ({ clicks }) =>
```

```
<Fragment>
```

```
<span className="clicks-count">{ clicks }</span>
```

```
</Fragment>
```

```
;
```

Presently our entire test suite is passing:

TAP variant 13

Hello segment

alright 1 Given a username: should Render a welcome to the right username.

ClickCounter segment

alright 2 Given a tick tally: should render the right number of snaps.

alright 3 Given a tick check: should render the right number of snaps.

1..3

tests 3

pass 3

alright

Time to test the catch. In the first place, include the test and watch it come up short (TDD style):

```
{
```

```
const $ = createCounter(0);
```

```
def generateReport():
```

```
    f1 = open(r"Report.txt", "w");
```

```
f1.write(Name.get()+","+Symptom1.get()+","+Symptom2.get()+","+Symptom3.get()+","+Symptom4.get()+","+Symptom5.get()+","+t1.get("1.0",END)+","+t2.get("1.0",END)+","+t3.get("1.0",END));
```

assert({should also load the component in the browser and see for yourself that the button works and the UI responds.

Implementing functional/e2e tests for React is the same as implementing them for any other framework. I won't go into them here, but check out TestCafe, TestCafe Studio and Cypress.io for e2e testing without the Selenium dance.

summarize:

- Find your Component Contract first
- Decide which constraints are worth testing and which aren't
- Prop types are not worth testing
- Inline styles are usually not worth testing
- The components you render and what props you give them are important to test
- Don't test things that are not the concern of your component

4.1 Type of Testing

4.1.1 Unit testing

When testing React parts, we will test both our desires for what is contained in the virtual dom just as what is reflected in the genuine dom. Unit testing alludes to testing singular pieces (or units, subsequently the name) of our code so we can be sure these particular bits of code fill in as we anticipate.

For instance, we have a couple of reducers as of now in our application. These reducers involve a solitary capacity that we can make attestations on under various situations. In React, Unit tests commonly don't require a program, can run extraordinarily rapidly (no composition to the DOM required), and the statements themselves are normally basic and succinct.

We'll for the most part focus on responding to the inquiry: with a given arrangement of sources of info (state and props), does the yield coordinate our desires for what should be in the virtual dom. For this situation, we're trying the rendering yield.

4.1.2 Functional testing

With utilitarian testing, we're centered around testing the conduct of our segment. For example, in the event that we have a route bar with a client login/logout catch, we can test our desires that:

- Given a signed in client, the navbar renders a catch with the text Logout
- Given no signed in client, the navbar renders a catch with the text Login

Practical tests as a rule keep running in disconnection (for example testing the segment usefulness without the remainder of the application).

4.1.3 Integration testing

At long last, the last kind of testing we'll take a gander at is incorporation trying. This kind of testing tests the whole administration of our application and endeavors to reproduce the experience an end-client would encounter when utilizing our application.

On the request of speed and effectiveness, reconciliation testing is amazingly moderate as it needs to run desires against a live, running program, whereas unit and utilitarian tests can run significantly quicker (particularly in Respond where the useful test is trying against the in-memory virtual dom as opposed to a real program render).

4.2 The Tools

How about we fix that by rendering the normal selector: We're going to utilize a testing library called jasmine to give an intelligible testing language and statements. To the extent test running, there is a general discussion around which test sprinter is the least demanding/most effective to work with, to a great extent between mocha and jest.

We're going to utilize Joke in our experience in testing with Respond as it's the official (take this while taking other factors into consideration) test sprinter. The majority of the code we'll be composing will be in Jasmine, so don't hesitate to utilize mocha, if it's your test library of decision.

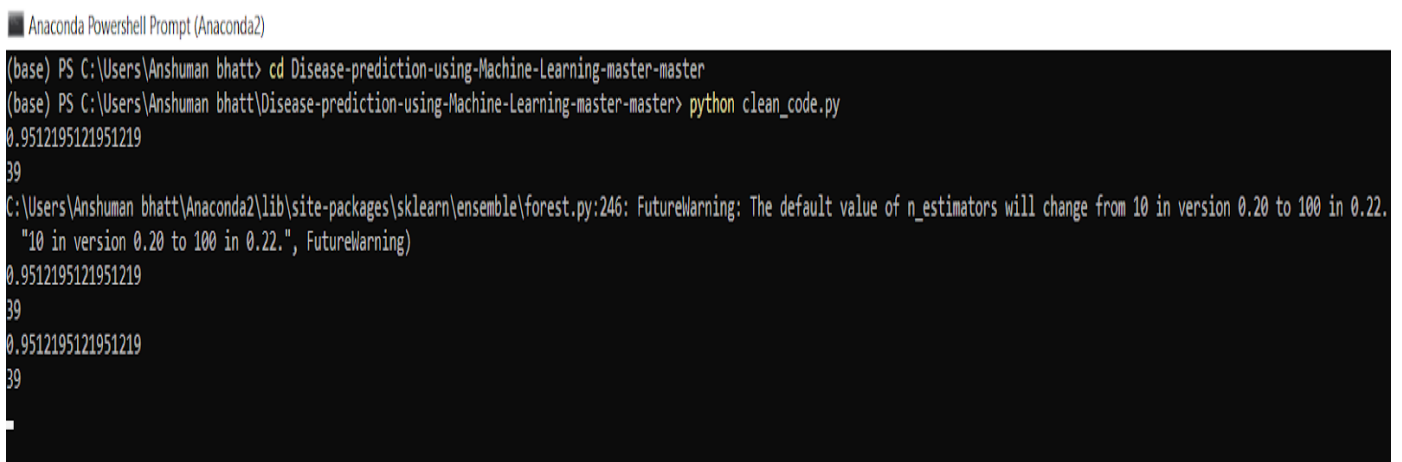
At long last, we'll utilize a library we can't live without called Enzyme which returns the enjoyment in utilitarian testing. Chemical gives some quite pleasant Respond testing utility capacities that make composing our declarations a snap.

4.3 Why Testing?

There are a few reasons which unmistakably explains to us as why Programming Testing is significant and what are the real things that we ought to consider while testing of any item or application. Programming testing is significant as a result of the accompanying reasons:

Programming testing is truly required to bring up the imperfections and mistakes that were made amid the improvement stages.

- 1). It's fundamental since it ensures the Client's unwavering quality and their fulfillment in the application.
- 2). It is critical to guarantee the Nature of the item. Quality item conveyed to the clients helps in picking up their certainty.
- 3). Testing is vital so as to give the offices to the clients like the conveyance of great item or programming application which requires lower upkeep cost and thus results into increasingly exact, predictable and solid outcomes.
- 4). Testing is required for a successful execution of programming application or item.
- 5). It's imperative to guarantee that the application ought not result into any disappointments since it very well may be over the top expensive later on or in the later phases of the improvement.
- 6). It's required to remain in the business.



```
Anaconda Powershell Prompt (Anaconda2)
(base) PS C:\Users\Anshuman bhatt> cd Disease-prediction-using-Machine-Learning-master-master
(base) PS C:\Users\Anshuman bhatt\Disease-prediction-using-Machine-Learning-master-master> python clean_code.py
0.9512195121951219
39
C:\Users\Anshuman bhatt\Anaconda2\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
0.9512195121951219
39
0.9512195121951219
39
```

Fig 4.4(a): Map-Track in Testing mode

```
Users component
  ✓ shows a list of users (11ms)

console.log src/__tests__/Users.test.js:17
  { data:
    [ { name: 'Kevin Mitnick' }, { name: 'Valentino Gagliardi' } ] }

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        0.151s, estimated 1s
Ran all test suites related to changed files.

Watch Usage: Press w to show more.█
```

Fig 4.4(b): Map-Track in Testing mode with mock-data

Chapter 5

Conclusion and Future Scope

In this toolkit a Machine learning Decision tree map algorithm by using structured and unstructured data from hospital. To the highest of gen, none of the current work attentive on together data types in the zone of remedial big data analytics. Compared to several typical calculating algorithms, the scheming accuracy of our proposed algorithm reaches 94.8% with

an regular speed which is quicker than that of the CNN-based unimodal disease risk prediction and produces report. The report consists of possibility of occurrences of diseases.

Different from many other systems it is able to both monitor and prediction. The diagnosis system of the system is able to predict the disease by using ML algorithms and the prediction results are based on the disease dataset instance. On the other hand, the system is very inexpensive for persons to use just by running it over their system. The experiment was carried out with the holdout test and the accuracy of the proposed system was 89% achieved with the Random forest. This project aims to predict the disease on the basis of the symptoms. The project is designed in such a way that the system takes symptoms from the user as input and produces output i.e. predict disease.

References

- [1] Vinitha S, Sweetlin S, Vinusha H and Sajini S “DISEASE PREDICTION USING MACHINE LEARNING” Computer Science & Engineering: An International Journal (CSEIJ), Vol.8, No.1, February 2018
- [2] S.Nandhini, Monojit Debnath, Anurag Sharma, Pushkar “Heart Disease Prediction using Machine Learning”, International Journal of Recent Engineering Research and Development