MACHINE LEARNING – 3 (SUPPORT VECTOR MACHINE, DECISION TREES)

A PROJECT REPORT

Submitted by

ANUSHKA AGRAWAL [17104032]
VATSAL GUPTA [17104060]
SAURAV KUMAR [17104065]
MANISHA RATHORE [17104033]
RAHUL MALHOTRA [17104046]

Under the guidance of

Ms. Adwitiya Sinha
(Assistant Professor, Department of Computer Science & IT)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHNOLOGY

in

INFORMATION TECHNOLOGY

Sector 62, Noida, Uttar Pradesh 201309

November 2020

ACKNOWLEDGEMENTS

We would like to express our deepest gratitude to our course faculties, Dr. Adwitiya Sinha and Dr. Megha Rathi, for their valuable guidance, consistent encouragement, timely help, and for providing us with an excellent atmosphere for doing our project. During the entire duration of our dissertation work, despite their busy schedule, they have extended cheerful and cordial support to us for completing this project work. We would also like to convey our sincere regards to all other faculty members of the department of CSE & IT, JIIT, who have bestowed their great effort and guidance at appropriate times. Our thanks and appreciations also go to people who have willingly helped in developing this project.

Contents

List of Tables

List of Figures

ABBREVIATIONS

SVM	Support Vector Machine
RBF	Radial Basis Function
EDA	Exploratory Data Analysis
CART	Classification and Regression Tree
EDA	Exploratory Data Analysis
HDD	Hard Disk Drive
SSD	Solid State Drive
SVR	Support Vector Regression
ID3	Iterative Dichotomiser 3

Chapter 1

INTRODUCTION

1.1 What is Classification?

In data science, classification refers to a supervised learning approach wherein a predictive model learns from past examples to anticipate the class label for a given instance of new data. Each instance is termed as an observation. Each observation has multiple attributes, which are known. The main goal is to identify which class the observation will fall into. Classes are also referred to as targets/labels or sometimes as categories. The data used for modeling classification models can be both - structured and unstructured. Examples of classification tasks include a) classifying whether or not a mail is spam, b) classifying whether it will rain the next day, etc. Primarily, there are four types of classification:

1. **Binary Classification:** This classification involves using a classification rule to approximate observations into either of the two-class labels. Binary classification uses Bernoulli probability distribution for the prediction. The algorithms for these classifications are Support Vector Machine (SVM), Linear Regression, K-Nearest Neighbor (KNN), Naive Bayes, among others [1].

2. **Multi-class Classification:** This classification has more than two class labels, but each observation can be categorized into only one of them. Multi-class classification uses Multinoulli probability distribution (also known as categorical distribution) for the prediction. The algorithms for this type of classification include Decision Trees, Random Forest, Gradient Boosting, Naive Bayes, among others [1].

3. **Multi-label Classification:** This classification has two or more class labels, and each input instance can belong to more than one class. This classification technique also uses Bernoulli probability distribution for making predictions [2].

4. **Imbalanced Classification:** This type of classification is required when the number of input examples across classes is not distributed equally. This can be seen as binary classification, with most input examples belonging to just one class. Performance metric measures are calculated by precision, F-measure, and recall [3].

1.2 What is Supervised Learning?

Supervised learning is a sub-branch of machine learning that involves making use of algorithms that study labeled data to establish the relationship between the input and output variables. In other words, supervised learning techniques enable us to define a mapping function so that we can predict the behavior of the output variable for newly presented input data. It is a powerful tool to classify and process data. Supervised learning is broadly classified into regression and classification problems.

- **Classification:** a problem where the output variable is a category like different colors red, yellow, green, etc. Example: Sentiment analysis and drug classification.
- **Regression:** a problem where the output variable is a real value like money, score, pounds. The model produces a numerical value. Example: predicting real estate prices [4].

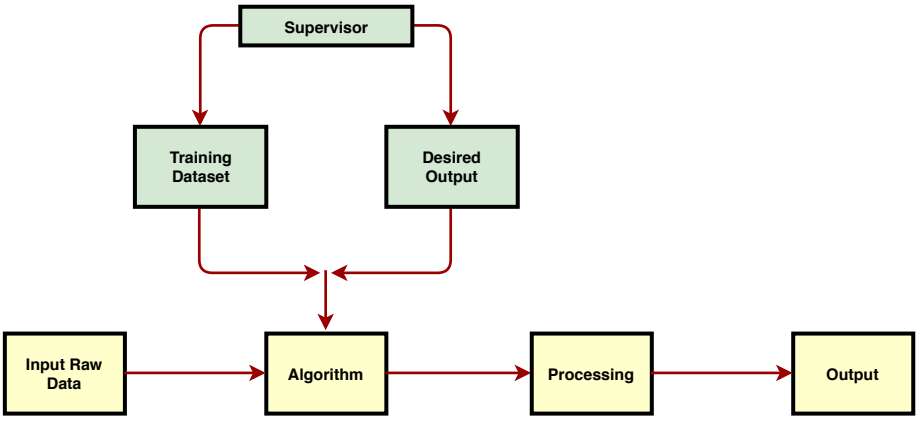


Figure 1.1: Supervised Learning

1.2.1 Examples of Supervised Learning Algorithms

- **Support Vector Machine (SVM):** Mainly used for classification problems. For example, image classification, financial performance comparison.
- **Linear Regression:** Mainly used for regression problems. For example, sales forecasting, risk assessment.
- **Random Forest:** Used for both regression and classification problems [5].
- **Decision Trees**
- **Logistic Regression**
- **K-Nearest Neighbours**

Chapter 2

SUPPORT VECTOR MACHINE (SVM)

2.1 Introduction to SVM

SVM is a supervised machine learning algorithm that finds a hyperplane or separating line that separates data of two classes. This algorithm is used for classification purposes. A support vector machine is an algorithm that takes the data as an input and outputs the hyperplane that separates two classes. The goal is to get a boundary from which separation between these classes is maximum. Steps involved in this algorithm:

1. Find the points which are close to hyperplane from both the classes. These points are called support vectors.
2. Calculate the distance between the hyperplane and support vectors. This distance is called the margin.
3. Get the hyperplane for which margin is maximized [6].

2.1.1 Types of SVM

- **Simple SVM (Linear SVM):** It is used to classify linearly separable data points [6]. Further elaboration of Linear SVM is included in section 2.3.2.
- **Kernel SVM (Non-linear SVM):** It is used when data points are not linearly separable. The main idea behind Kernel SVM is adding another dimension to make the data separable. This can be done be drawing a separating hyperplane [6]. Non-linear SVM is explained further in section 2.3.3.

2.1.2 Advantages of SVM

1. It works well even when the number of SVM features is much larger than the number of instances.
2. SVM is available with almost all data analytics toolsets.
3. It can work on a huge feature space.
4. It is easy to understand and implement [6].

2.1.3 Disadvantages of SVM

1. Training a SVM model is a time-consuming process when the data set is large.
2. If there is much noise in the data set, then the performance of SVM classifiers gets impeded.
3. SVM classifiers make use of a hyperplane to categorize data points. Therefore, classification done using SVM has no probabilistic explanation [7].

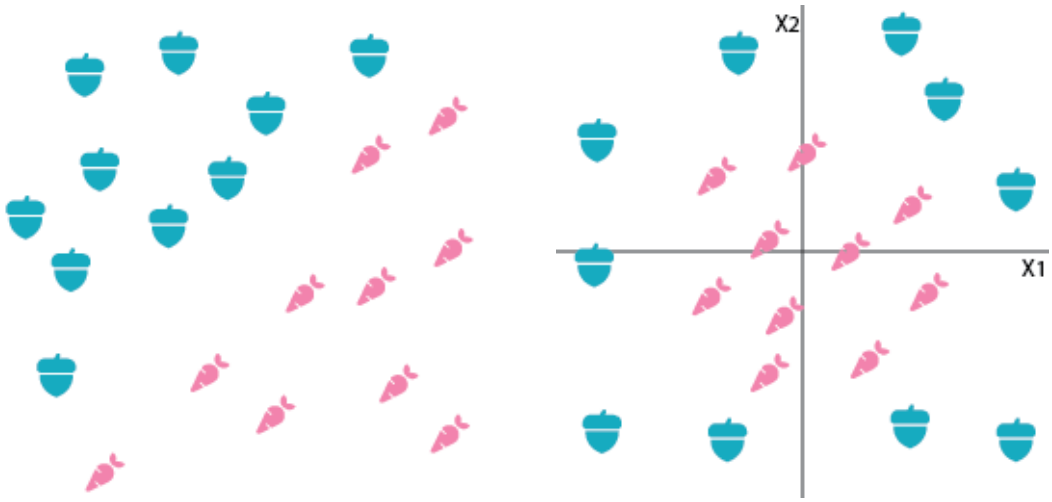
2.2 SVM Applications

- **Face detection:** Class labels for face detection in SVM are - "face" and "non-face". SVM classifier 1) extracts all the features from the picture, 2) makes a square boundary around the image, and then 3) classifies the picture as a face, or non-face [8].
- **Text classification:** We apply SVM for differentiating among the handwriting of people. The dataset comprises photos of numerous alphabets, sentences, and paragraphs written in each kind of handwriting. This can also be used for differentiating the handwriting of humans from computer alphabets [9].
- **Stenography detection:** SVM is also used for differentiating unadulterated images from adulterated ones. This application is used in security-based organizations. Encrypted messages are transmitted secretly and are hidden under the images. We encrypt the messages in high-decision images where extra pixels are extra tough to find out. SVM is then used to segregate the pixels in the dataset and examine the image [10].
- **Bioinformatics:** Cancer classification and protein classification comes under this application. We use SVM for figuring out the type of genes, sufferers of the idea of genes, and different organic and biological problems.

2.3 Linear and Non-linear SVM

2.3.1 Linear Separability

A dataset in n dimensions is called linearly separable if there exists a hyperplane for the n-dimensional space which can separate the data points such that data points belonging to one class lie on one side of the hyperplane and the data points belonging to the other class lie on another side of the hyperplane [11].



(a) Linearly separable dataset **(b)** Linearly inseparable dataset

Figure 2.1: Linear Separability

2.3.2 Linear SVM

Linear SVMs try to find a hyperplane that can separate the data points belonging to two or more different classes in a linearly separable dataset [12].

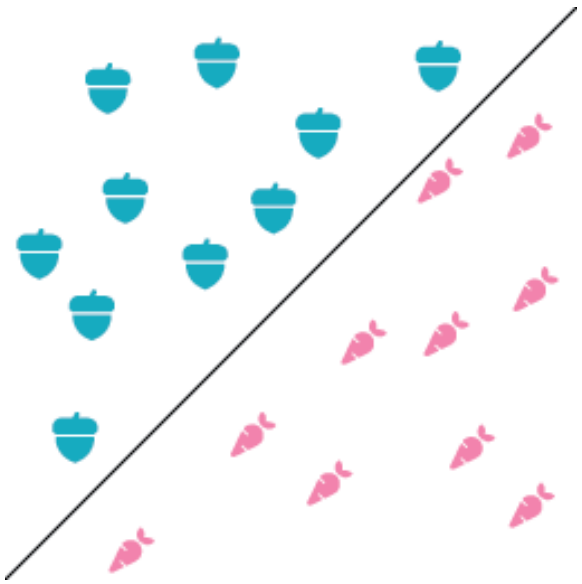
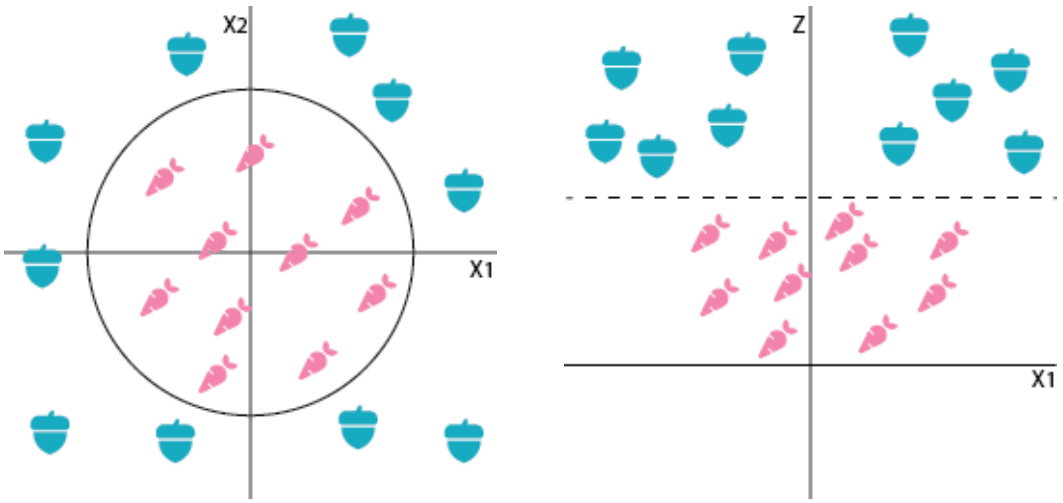


Figure 2.2: Hyperplane Dividing 2 Different Classes

2.3.3 Non Linear SVM

When the dataset is not linearly separable, Linear SVMs cannot find a hyperplane which could divide the data points into different classes [13]. Non-linear SVMs make use of kernel trick to introduce more dimensions in which the hyperplane could divide the data points into their respective different classes [14].



(a) Non-linear plane which divides the dataset into 2 different classes

(b) Kernel trick introduces new dimension (z = $x_1^2 + x_2^2$)

Figure 2.3: Non Linear SVM

2.3.4 Differences

Table 2.1: Difference between Linear & Non Linear SVM

Linear SVM	**Non-Linear SVM**
1. Separates linearly separable dataset.	1. Separates dataset which is not linearly separable.
2. Dataset is divided into classes with a hyperplane.	2. Dataset is divided using kernel trick.
3. No dataset mapping is required.	3. Dataset is mapped into a higher dimensional space for classification.

2.3.5 Kernel Functions

Kernel functions map data points from one dataset to a higher dimensional space. Examples of kernel functions:

Polynomial Kernel

Polynomial kernel is very commonly used in image processing.

$$k\left(\mathbf{x_i}, \mathbf{x_j}\right) = \left(\mathbf{x_i} \cdot \mathbf{x_j} + 1\right)^d \tag{2.1}$$

Gaussian Kernel

Gaussian kernel is used for general-purpose processing where much information about data is not available.

$$k(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \tag{2.2}$$

Gaussian Radial Basis Function

Like Gaussian kernel, Gaussian Radial Basis Function (RBF) is used for general-purpose processing. The main difference between the two is that regularisation is applied in Gaussian RBF, but not in Gaussian kernel. A regularised RBF network typically uses a penalty based on the squared norm of the weights.

$$k\left(\mathbf{x_i}, \mathbf{x_j}\right) = \exp\left(-\gamma \left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2\right) \tag{2.3}$$

Hyperbolic Tangent Kernel (Sigmoid Kernel)

The Hyperbolic Tangent kernel, also known as the Multilayer Perceptron (MLP) kernel or the Sigmoid kernel, is commonly used in neural networks.

$$k\left(\mathbf{x_i}, \mathbf{x_j}\right) = \tanh\left(\kappa \mathbf{x_i} \cdot \mathbf{x_j} + c\right) \tag{2.4}$$

Bessel Kernel

Bessel function is used to remove the cross term in mathematical functions.

$$k(x,y) = \frac{J_{v+1}(\sigma\|x-y\|)}{\|x-y\|^{-n(v+1)}} \tag{2.5}$$

ANOVA Kernel

ANOVA kernel, another radial basis function kernel, is very commonly used for solving regression problems.

$$k(x,y) = \sum_{k=1}^{n} \exp\left(-\sigma\left(x^k - y^k\right)^2\right)^d \tag{2.6}$$

Spline Kernel

Spline kernel, given by Steve Gunn in [15], is a piecewise cubic polynomial function, commonly used in text categorization problems [16].

$$k(x,y) = 1 + xy + xy\min(x,y) - \frac{x+y}{2}\min(x,y)^2 + \frac{1}{3}\min(x,y)^3 \tag{2.7}$$

2.4 SVM Classification and Regression

SVMs are machine learning models widely used in classification problems, where we have to assign a class to a data point based on its feature. SVMs can also be used for regression problems to predict a real number based on the input provided.

2.4.1 How SVM works for Classification Problems?

For classification problems, SVM algorithms try to find the optimal hyperplane which categorizes the data points. The optimal hyperplane is one with maximum margin.

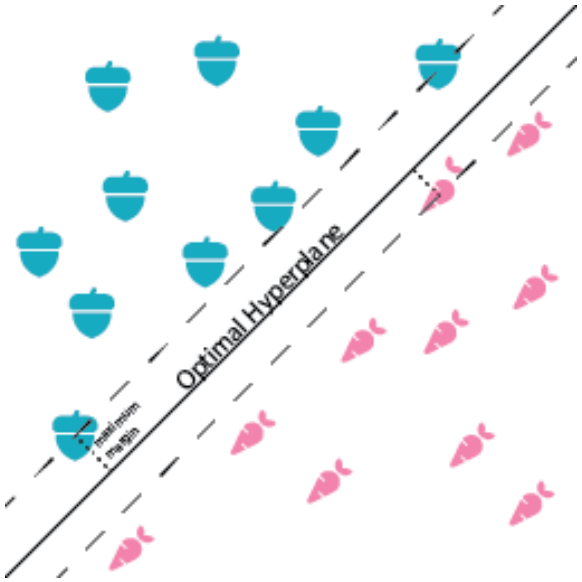


Figure 2.4: Optimal Hyperplane - SVM Classification

Parameters Involved

1. **Kernel:** It tells the SVM algorithm which kernel trick to use to handle non linearly separable datasets.
2. **Penalty Parameter (C):** It tells the SVM algorithm how much significance to give to misclassified data points. Larger C values create more nuanced boundaries for the trained models.
3. **Gamma:** Gamma tells the SVM algorithm how much significance datapoints away from the boundary have on the shape of the boundary.

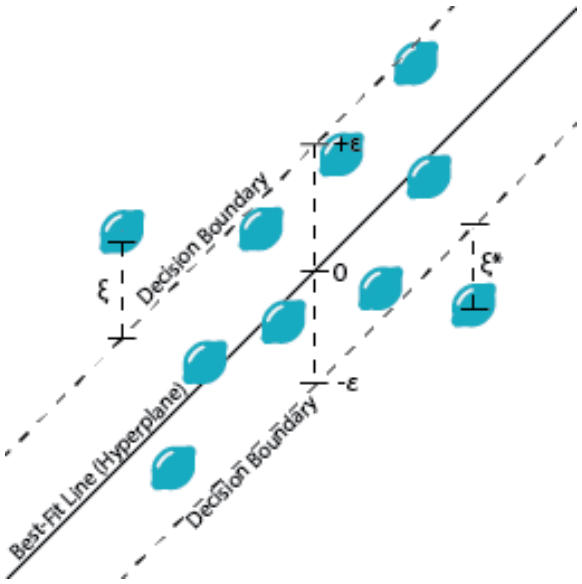


Figure 2.5: SVM for Regression

2.4.2 How SVM works for Regression Problems?

For regression problems, SVM algorithms consider the datapoints lying within the decision boundary and creates the hyperplane, which acts as the best-fit line for the problem.

2.5 Python Implementation of SVM

2.5.1 SVM Classification

Classification Dataset

The classification dataset used in our SVM implementation contains a total of 6 attributes, 4 of which are used for the purpose of classification, i.e. there are a total of 4 parameters. Attributes:

1. **Id:** Integer to uniquely identify each record
2. **Len1 - Len4:** Length of different features
3. **Output:** The class label

Code

See Appendix B.1.

Output

2.5.2 SVM Regression

Regression Dataset

The regression dataset used in our implementation consists of information about businesses and their output in a particular month. There are a total of 43 attributes in our dataset:

1. **Id:** Integer to uniquely identify each record

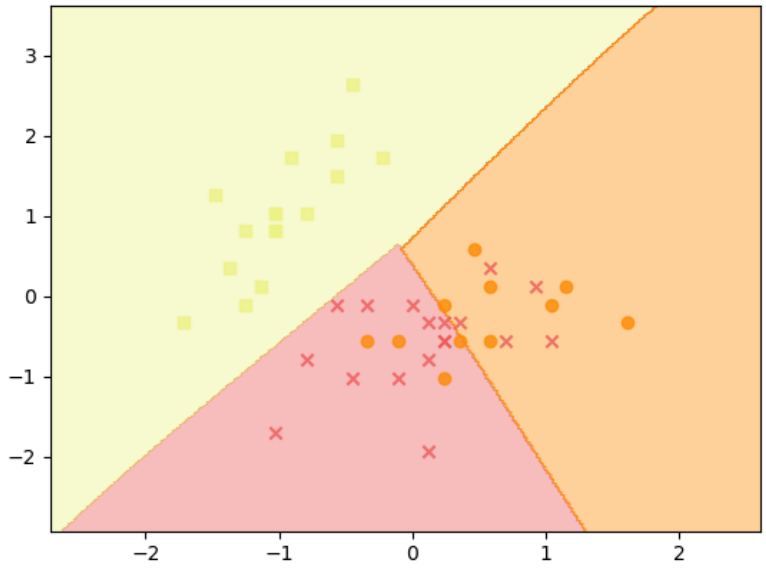


Figure 2.6: Lines dividing the region into different classes and plotted along data points. Class 1 (square), Class 2 (cross) & Class 3 (circle).

2. **Start:** Date of start of business
3. **Place:** Location of the business
4. **Group:** Weather city is located in a Big City or not
5. **Category:** The category the location belongs to
6. **P1-P37:** Score based on a particular parameter
7. **Output:** Output of the business in that month

Code

See Appendix B.2.

Output

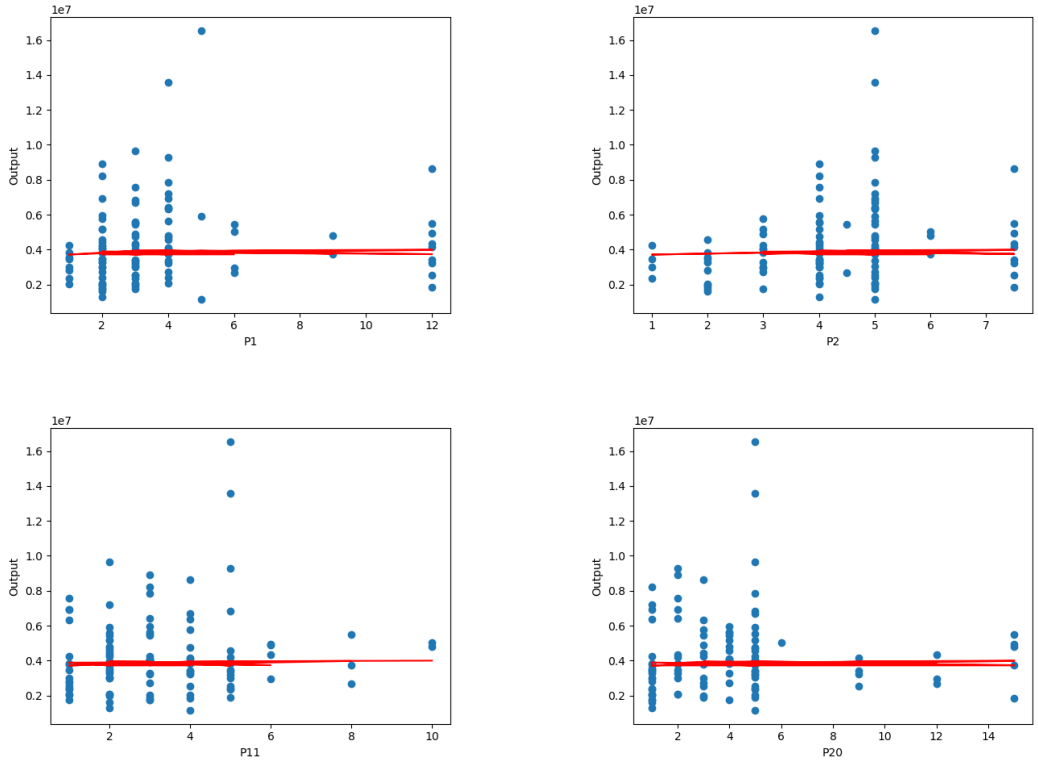


Figure 2.7: Regression line (red) and data points (blue) plotted against parameters: a) P1, b) P2, c) P11, d) P20

Chapter 3

DECISION TREES

3.1 Introduction to Decision Trees

A decision tree is a flowchart like structure based on supervised learning that can be used for classification and prediction. A decision tree consists of a root node and several interior nodes & terminal nodes. Each internal node denotes a test on an attribute, each branch represents an outcome of the trial, and each leaf node indicates the class label. Therefore each branch of the tree represents a possible decision, occurrence, or reaction. The flexibility of the decision tree classifier makes it attractive for a wide range of applications, either for improving the classifier performance in general (maximizing accuracy while minimizing computational requirements) or treating special applications in which multilevel decision logic is the only practical approach [17].

3.1.1 Advantages of Decision Tree Algorithm

1. It is easy to implement and visualize.
2. Data preparation is simpler as compared to other algorithms (scaling not required).
3. Can handle both qualitative and quantitative data.
4. Non-Linear parameters do not affect model performance.

3.1.2 Disadvantages of Decision Tree Algorithm

1. When results are calculated for a specific instance rather than a general trend, the model may over-fit.
2. The model can get unstable due to small variations in data.
3. Complicated trees tend to have a low bias which makes it difficult to work with new data.

3.1.3 Applications of Decision Tree

- **Bioinformatics:** Decision tree-based classifiers are applied in cancer classification, including breast cancer, central nervous system embryonic tumour, colon tumour, leukaemia, lung cancer, ovarian cancer, pancreatic cancer, and prostate cancer. It is also widely used in genomics classification to identify protein-coding regions in the genome sequences [18].
- **Stock Market Future Trends:** Since a Decision tree can handle a dataset with quantitative and qualitative data, it is perfect for a stock recommendation and trade assistance [19].
- **Fraudulent Statement Detection:** Earlier fraud financial statements were identified using statistical methods. The decision tree proved to be very useful in identifying FFS with an accuracy of more than 90% [19].

3.2 Types of Decision Tree Algorithm

- **Classification and Regression Tree (CART):** It is a dynamic supervised learning algorithm which may produce classification and regression tree as per the nature of the dependent variable. CART models use Gini Index as the splitting criteria. For more information on Gini Index, refer to section 3.3.3
- **Iterative Dichotomiser 3 (ID3):** The algorithm uses information gain to determine which attribute should be used to classify the current subset of the data [20].
- **C4.5:** This is even better as compared to the ID3 algorithm. It uses either information gain or gain ratio to decide the classifying attribute. It can handle both continuous and missing value attributes [20].

3.3 Splitting Criteria

3.3.1 Entropy

Entropy is the measure of randomness or uncertainty in a dataset. Higher entropy implies that more classes would be required to fit the data [21].

$$E(S) = \sum_{i=1}^{c} -p_i \log_2 p_i \tag{3.1}$$

where S is the subset of training examples and c is the total number of classes.

3.3.2 Information Gain

Information Gain is a measure of the decrease in entropy after the dataset is split. It helps in deciding which node should be considered as the decision node. A high information gain implies a more elevated position for the node in the decision tree.

$$\text{IG}\,(S, A) = \text{Entropy}\,(S) - \Sigma\,((|S_v|\,/|S|) * \text{Entropy}\,(S_v)) \tag{3.2}$$

where S_v is the set of rows in S for which the feature column A has value v, $|S_v|$ is the number of rows in S_v and likewise $|S|$ is the number of rows in S.

3.3.3 Gini Index

Gini index is the probability that a specific feature is incorrectly classified when selected randomly. A lower value of Gini implies a better choice of feature for classifying.

$$\text{Gini} = 1 - \sum_{i=1}^{c} (p_i)^2 \tag{3.3}$$

3.3.4 Gain Ratio

In the original paper by Ross Quinlan [22], Information Gain Ratio, commonly referred to as just Gain Ratio, is defined as the ratio of information gain to the intrinsic information. Gain Ratio is a modified version of the Information Gain measure. The main motive behind the calculation of Gain Ratio is to reduce a bias towards multi-valued attributes by taking the intrinsic information of a split into account (information about the class in disregarded). Fundamentally, Gain Ratio takes into consideration, both the number and the size of branches while choosing an attribute.

$$\text{GR}\,(S, A) = \frac{\text{Gain}\,(S, A)}{\text{SplitInfo}\,(S, A)} \tag{3.4}$$

3.4 Python Implementation of Decision Trees

3.4.1 Dataset Description

The chosen dataset is based on the overall achievement of students. The dataset contains four features (Academics, Speaking Skills, Creative Skills, Sports) of 20 samples of three overall performance remarks → Excellent, Average, and Poor. The students are marked in each parameter, according to which the overall achievement is calculated.

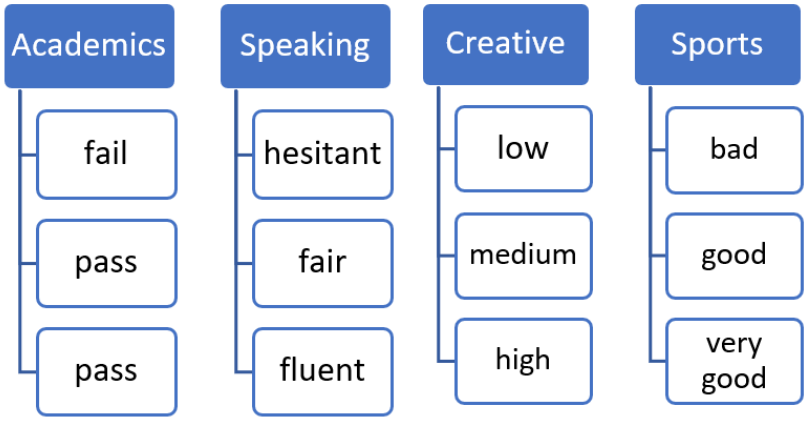


Figure 3.1: Dataset

Independent Variables

- Academics
- Speaking Skills
- Creative Skills
- Sports

Dependent Variable

- Overall Achievement

Table 3.1: Dataset for ID3 and C4.5 Implementation

Academics	Speaking	Creative	Sports	Overall Achievement
Pass	Fair	Medium	Bad	AVERAGE
Distinction	Fluent	Medium	Very Good	EXCELLENT
Pass	Hesitant	Low	Bad	POOR
Pass	Fair	Medium	Good	AVERAGE
Distinction	Fluent	High	Good	EXCELLENT
Pass	Hesitant	Medium	Bad	POOR
Pass	Hesitant	Medium	Bad	POOR
Pass	Fair	Medium	Very Good	AVERAGE
Pass	Hesitant	Low	Bad	POOR
Pass	Hesitant	High	Good	AVERAGE
Pass	Fair	Medium	Bad	AVERAGE
Fail	Hesitant	High	Very Good	EXCELLENT
Distinction	Fluent	High	Good	EXCELLENT
Distinction	Hesitant	Medium	Good	EXCELLENT
Distinction	Hesitant	Low	Good	AVERAGE
Pass	Fair	Low	Bad	POOR
Fail	Hesitant	Medium	Bad	POOR
Fail	Hesitant	Low	Bad	POOR
Pass	Fair	High	Bad	AVERAGE
Distinction	Fluent	Medium	Very Good	EXCELLENT

3.4.2 Exploratory Data Analysis (EDA)

The Exploratory Data Analysis (EDA) reveals that 30% students are graded as 'Excellent' while 35% students are graded under 'Average' and rest 35% are categorised under 'Poor' performance.

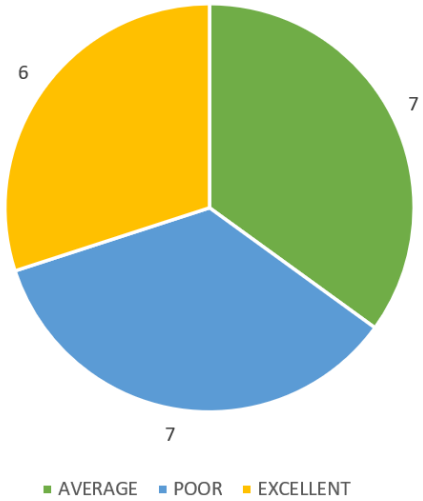


Figure 3.2: Exploratory Data Analysis

3.4.3 ID3

See Appendix B.3.

3.4.4 C4.5

See Appendix B.4.

3.4.5 Analysis & Results

Figure 3.3 is the decision tree formed by implementing ID3 and C4.5 algorithms on the given dataset. Depth of the tree=4. Each node contains several characteristics of the node example, the entropy, which decides to split the data and the number of data samples under that node. The leaf nodes are red, orange, and yellow coloured, denoting 'Excellent', 'Average' and 'poor' respectively as the overall achievement.

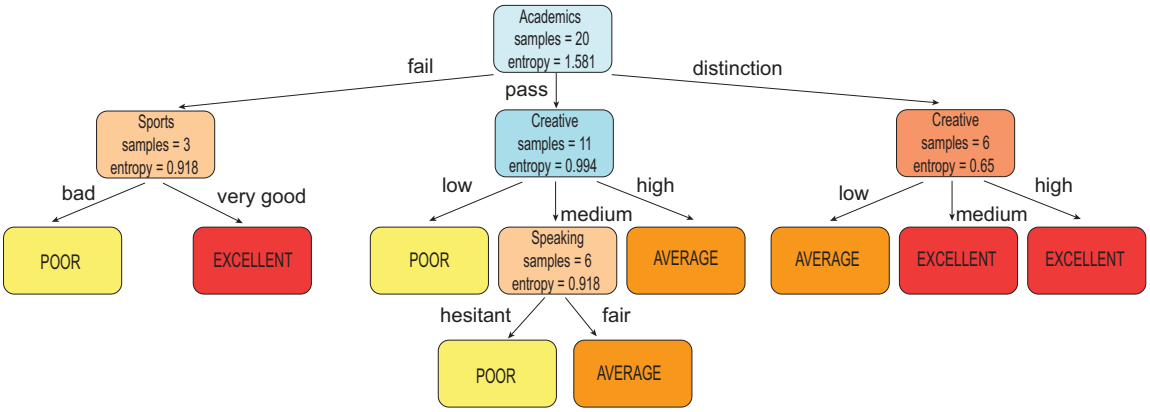


Figure 3.3: Final Decision Tree

The root node is 'Academics', which shows that 'Academics' has the highest gain amongst all other attributes, which means that the first split is done on the feature 'Academics'. If a student has failed in Academics, then his grade against 'Sports' is considered for deciding the Overall Achievement; else, 'Creative' is regarded as the next attribute for splitting. Similarly, further splitting of data is done on the basis of highest gain in case of ID3 and highest gain ratio in the case of C4.5 algorithms.

Let sample input data be:

Table 3.2: Sample Input Example

Academics	Speaking	Creative	Sports	Overall Achievement
Pass	Hesitant	Medium	Bad	POOR

According to the decision tree in the figure 3.3, we will first compare Academics then coming on the next level of the decision tree; our data qualifies under creative $\rightarrow$ medium. Then on the last level under speaking $\rightarrow$ hesitant, the given sample takes the decision as 'Poor'.

3.5 Solved Numerical

3.5.1 ID3

- **Step 1:** Calculate the entropy of the dependent attribute using formula 3.1. In our case, c = 3. Possible classes are: 1) poor, 2) average, 3) excellent.

$$\begin{aligned}\text{Entropy}\,(S) &= -\text{p}\,(\text{poor})\log\left(\text{p}\,(\text{poor})\right) - \text{p}\,(\text{average})\log\left(\text{p}\,(\text{average})\right) \\ &\quad - \text{p}\,(\text{excellent})\log\left(\text{p}\,(\text{excellent})\right) \\ &= -\frac{7}{20}\log\left(\frac{7}{20}\right) - \frac{7}{20}\log\left(\frac{7}{20}\right) - \frac{6}{20}\log\left(\frac{6}{20}\right) \\ &= 1.581\end{aligned} \tag{3.5}$$

- **Step 2:** Calculate the entropy of each independent variable.

Table 3.3: Academics - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Fail	2	0	1	0.9182
Pass	5	6	0	0.994
Distinction	0	1	5	0.65

$$\begin{aligned}\text{Entropy}\,(\text{academics = "fail"}) &= -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - 0 - \frac{1}{3}\log_2\left(\frac{1}{3}\right) \\ &= 0.918\end{aligned} \tag{3.6}$$

$$\begin{aligned}\text{Entropy}\,(\text{academics = "pass"}) &= -\frac{5}{11}\log_2\left(\frac{5}{11}\right) - \frac{6}{11}\log_2\left(\frac{6}{11}\right) - 0 \\ &= 0.994\end{aligned} \tag{3.7}$$

$$\text{Entropy (academics = "distinction")} = 0 - \frac{1}{6}\log_2\left(\frac{1}{6}\right) - \frac{5}{6}\log_2\left(\frac{5}{6}\right) = 0.65 \tag{3.8}$$

Table 3.4: Speaking - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	6	2	2	1.371
Fair	1	5	0	0.65
Fluent	0	0	4	0

$$\text{Entropy (speaking = "hesitant")} = -\frac{6}{10}\log_2\left(\frac{6}{10}\right) - \frac{2}{10}\log_2\left(\frac{2}{10}\right) - \frac{2}{10}\log_2\left(\frac{2}{10}\right) = 1.371 \tag{3.9}$$

$$\text{Entropy (speaking = "fair")} = -\frac{1}{6}\log_2\left(\frac{1}{6}\right) - \frac{5}{6}\log_2\left(\frac{5}{6}\right) - 0 = 0.65 \tag{3.10}$$

$$\text{Entropy (speaking = "fluent")} = 0 - 0 - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0 \tag{3.11}$$

Table 3.5: Creative - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Low	4	1	0	0.722
Medium	3	4	3	1.571
High	0	2	3	0.971

$$\text{Entropy (creative = "low")} = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right) - 0 = 0.722 \tag{3.12}$$

$$\text{Entropy (creative = "medium")} = -\frac{3}{10}\log_2\left(\frac{3}{10}\right) - \frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{3}{10}\log_2\left(\frac{3}{10}\right) = 1.571 \tag{3.13}$$

$$\begin{aligned}\text{Entropy (creative = "high")} &= 0 - \frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) \\ &= 0.971\end{aligned} \tag{3.14}$$

Table 3.6: Sports - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Bad	7	3	0	0.881
Good	0	3	3	1
Very Good	0	1	3	0.811

$$\begin{aligned}\text{Entropy (sports = "bad")} &= -\frac{7}{10}\log_2\left(\frac{7}{10}\right) - \frac{3}{10}\log_2\left(\frac{3}{10}\right) - 0 \\ &= 0.881\end{aligned} \tag{3.15}$$

$$\begin{aligned}\text{Entropy (sports = "good")} &= 0 - \frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) \\ &= 1\end{aligned} \tag{3.16}$$

$$\begin{aligned}\text{Entropy (sports = "very good")} &= 0 - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) \\ &= 0.811\end{aligned} \tag{3.17}$$

- **Step 3:** Calculate the average information of each independent variable using the formula given below:

$$\text{Avg. Info.} = \Sigma\left(\left(|S_v|/|S|\right) * \text{Entropy}\left(S_v\right)\right) \tag{3.18}$$

where S_v is the set of rows in S for which the feature column A has value v, $|S_v|$ is the number of rows in S_v and likewise $|S|$ is the number of rows in S.

$$\begin{aligned}I\left(\text{academics}\right) = &\left(\left(\frac{2+0+1}{20} * 0.918\right)\right) + \left(\left(\frac{5+6+0}{20} * 0.994\right)\right) \\ &+ \left(\left(\frac{0+1+5}{20} * 0.65\right)\right) \\ &= 0.8794\end{aligned} \tag{3.19}$$

$$
\begin{aligned}
I\left(\text{speaking}\right) = & \left(\left(\frac{6+2+2}{20} * 1.371\right)\right) + \left(\left(\frac{1+5+0}{20} * 0.65\right)\right) \\
& + \left(\left(\frac{0+0+4}{20} * 0\right)\right) \\
& = 0.8805
\end{aligned}
\tag{3.20}
$$

$$
\begin{aligned}
I\left(\text{creative}\right) = & \left(\left(\frac{4+1+0}{20} * 0.722\right)\right) + \left(\left(\frac{3+4+3}{20} * 1.571\right)\right) \\
& + \left(\left(\frac{0+2+3}{20} * 0.971\right)\right) \\
& = 1.20875
\end{aligned}
\tag{3.21}
$$

$$
\begin{aligned}
I\left(\text{sports}\right) = & \left(\left(\frac{7+3+0}{20} * 0.881\right)\right) + \left(\left(\frac{0+3+3}{20} * 1\right)\right) \\
& + \left(\left(\frac{1+3+0}{20} * 0.811\right)\right) \\
& = 0.9027
\end{aligned}
\tag{3.22}
$$

- **Step 4:** Calculate the gain of each independent variable using the formula 3.2

$$
\begin{aligned}
\text{Gain}\left(\text{academics}\right) &= E\left(S\right) - I\left(academics\right) \\
&= 1.581 - 0.8794 = 0.7016
\end{aligned}
\tag{3.23}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{speaking}\right) &= E\left(S\right) - I\left(speaking\right) \\
&= 1.581 - 0.8805 = 0.7005
\end{aligned}
\tag{3.24}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{creative}\right) &= E\left(S\right) - I\left(creative\right) \\
&= 1.581 - 1.20875 = 0.3722
\end{aligned}
\tag{3.25}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{sports}\right) &= E\left(S\right) - I\left(sports\right) \\
&= 1.581 - 0.9027 = 0.6783
\end{aligned}
\tag{3.26}
$$

- **Step 5:** Choose the attribute with the highest gain and then repeat steps 1 to 4 for the dataset formed with the chosen attribute.

  Since "Academics" has the highest gain, we choose it as the root node.

Table 3.7: Deciding the Root Node

Attributes	**Gain**
Academics	**0.7016**
Speaking	0.7005
Creative	0.3722
Sports	0.6783

Now, repeat the steps 1-4 again. Total number of data points with academics → distinction = 6.

$$\begin{aligned}\text{Entropy}\left(S_{\text{distinction}}\right) &= -\frac{5}{6}\log_2\left(\frac{5}{6}\right) - \frac{1}{6}\log_2\left(\frac{1}{6}\right) - 0 \\ &= 0.65\end{aligned} \tag{3.27}$$

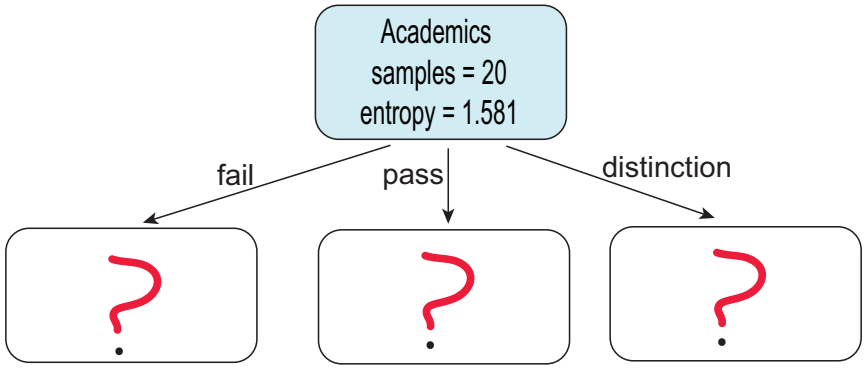


Figure 3.4: Root Node of the Decision Tree

Table 3.8: Speaking - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	0	1	1	1
Fair	0	0	0	0
Fluent	0	0	4	0

$$\text{Entropy}\left(\text{speaking = "hesitant"}\right) = 0 - \frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1 \tag{3.28}$$

$$\text{Entropy}\left(\text{speaking = "fair"}\right) = 0 \tag{3.29}$$

$$\text{Entropy}\left(\text{speaking = "fluent"}\right) = -\frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0 \tag{3.30}$$

$$I\left(\text{speaking}\right) = \left(\frac{2}{6} * 1\right) = 0.333 \tag{3.31}$$

$$\begin{aligned}\text{Gain (speaking)} &= E\left(S_{distinction}\right) - I\left(speaking\right) \\ &= 0.65 - 0.33 = 0.32\end{aligned} \tag{3.32}$$

Table 3.9: Creative - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Low	0	1	0	0
Medium	0	0	3	0
High	0	0	2	0

$$\text{Entropy (creative = "low")} = 0 \tag{3.33}$$

$$\text{Entropy (creative = "medium")} = 0 \tag{3.34}$$

$$\text{Entropy (creative = "high")} = 0 \tag{3.35}$$

$$I\left(\text{creative}\right) = 0 \tag{3.36}$$

$$\begin{aligned}\text{Gain (creative)} &= E\left(S_{distinction}\right) - I\left(creative\right) \\ &= 0.65 - 0 = 0.65\end{aligned} \tag{3.37}$$

Table 3.10: Sports - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	0	0	0	0
Good	0	1	3	0.811
Very Good	0	0	2	0

$$\text{Entropy (sports = "bad")} = 0 \tag{3.38}$$

$$\text{Entropy (sports = "good")} = 0 - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{4}\log_2\left(\frac{3}{4}\right) = 0.811 \tag{3.39}$$

$$\text{Entropy (sports = "very good")} = 0 \tag{3.40}$$

$$I\left(\text{sports}\right) = \left(\frac{4}{6} * 0.811\right) = 0.540 \tag{3.41}$$

$$\begin{aligned}\text{Gain (sports)} &= E\left(S_{distinction}\right) - I\left(sports\right) \\ &= 0.65 - 0.540 = 0.11\end{aligned} \tag{3.42}$$

Table 3.11: Deciding the Node after Academics→"Distinction"

Attributes	Gain
Speaking	0.32
Creative	**0.65**
Sports	0.11

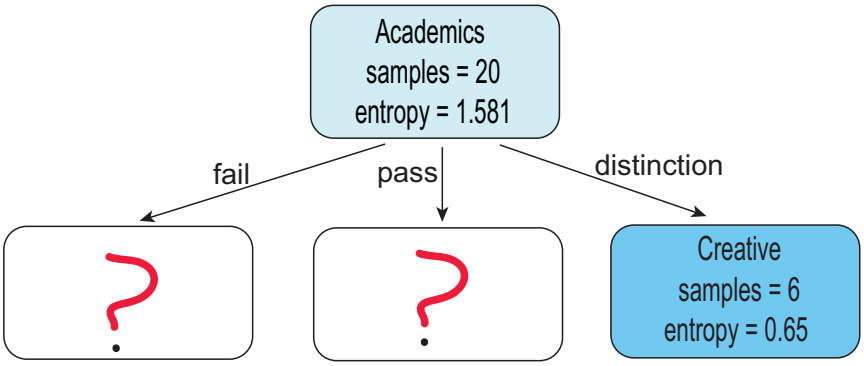


Figure 3.5: Node under Academics→"Distincton"

Next node will be "Creative":

Now, for Academics→"Pass", total number of data points = 11.

$$\text{Entropy}\,(S_{\text{pass}}) = 0.994 \tag{3.43}$$

Table 3.12: Speaking - For Academics→"pass"

	Poor	Average	Excellent	Entropy
Hesitant	4	1	0	0.722
Fair	1	5	0	0.65
Fluent	0	0	0	0

$$I\,(\text{speaking}) = \left(\frac{5}{11} * 0.722\right) + \left(\frac{6}{11} * 0.65\right) = 0.6827 \tag{3.44}$$

$$\text{Gain}\,(\text{speaking}) = 0.3113 \tag{3.45}$$

Table 3.13: Creative - For Academics→"pass"

	Poor	Average	Excellent	Entropy
Low	3	0	0	0
Medium	2	4	0	0.918
High	0	2	0	0

$$I\,(\text{creative}) = \left(\frac{6}{11} * 0.918\right) = 0.5 \tag{3.46}$$

$$\text{Gain}\,(\text{creative}) = 0.494 \tag{3.47}$$

Table 3.14: Sports - For Academics→"pass"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	5	3	0	0.954
Good	0	2	0	0
Very Good	0	1	0	0

$$I\,(\text{sports}) = \left(\frac{8}{11} * 0.954\right) = 0.6938 \tag{3.48}$$

$$\text{Gain}\,(\text{sports}) = 0.3 \tag{3.49}$$

Table 3.15: Deciding the Node after Academics→"Pass"

Attributes	**Gain**
Speaking	0.3113
Creative	**0.494**
Sports	0.30

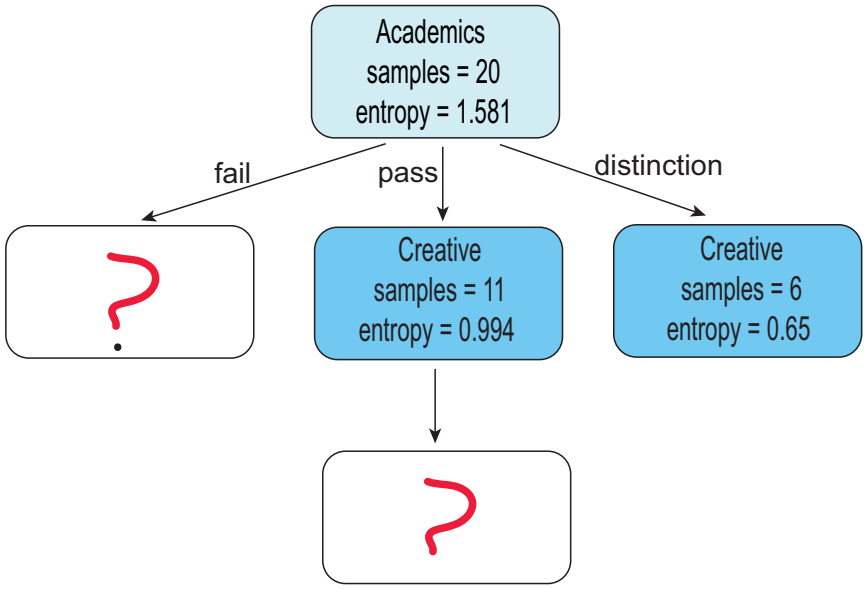


Figure 3.6: Node under Academics→"Pass"

Now, for Academics→"Fail", total number of data points = 3.

$$\text{Entropy}\,(S_{\text{fail}}) = 0.918 \tag{3.50}$$

$$I\,(\text{speaking}) = \left(\frac{3}{3} * 0.918\right) = 0.918 \tag{3.51}$$

Table 3.16: Speaking - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	2	0	1	0.918
Fair	0	0	0	0
Fluent	0	0	0	0

$$\text{Gain}\,(\text{speaking}) = 0 \tag{3.52}$$

Table 3.17: Creative - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Low	1	0	0	0
Medium	1	0	0	0
High	0	0	1	0

$$I\,(\text{creative}) = 0 \tag{3.53}$$

$$\text{Gain}\,(\text{creative}) = 0.918 \tag{3.54}$$

Table 3.18: Sports - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	2	0	0	0
Good	0	0	0	0
Very Good	0	0	1	0

$$I\,(\text{sports}) = 0 \tag{3.55}$$

$$\text{Gain}\,(\text{sports}) = 0.918 \tag{3.56}$$

Table 3.19: Deciding the Node after Academics→"Fail"

Attributes	**Gain**
Speaking	0
Creative	0.918
Sports	**0.918**

For the last branch of the root node, we have "Sports" as the next node. See figure 3.7. Similarly, we continue for all the nodes, we finally get figure 3.3.

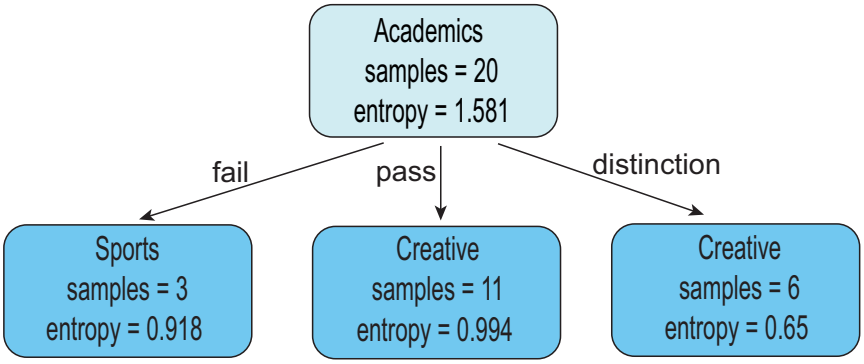


Figure 3.7: Node under Academics→"Fail"

3.5.2 C4.5

- **Step 1:** Calculate the entropy of the dependent attribute using formula 3.1. In our case, c = 3. Possible classes are: 1) poor, 2) average, 3) excellent.

$$\begin{aligned}\text{Entropy}\,(S) &= -\text{p}\,(\text{poor})\log\left(\text{p}\,(\text{poor})\right) - \text{p}\,(\text{average})\log\left(\text{p}\,(\text{average})\right) \\ &\quad - \text{p}\,(\text{excellent})\log\left(\text{p}\,(\text{excellent})\right) \\ &= -\frac{7}{20}\log\left(\frac{7}{20}\right) - \frac{7}{20}\log\left(\frac{7}{20}\right) - \frac{6}{20}\log\left(\frac{6}{20}\right) \\ &= 1.581\end{aligned} \tag{3.57}$$

- **Step 2:** Calculate the entropy of each independent variable.

Table 3.20: Academics - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Fail	2	0	1	0.9182
Pass	5	6	0	0.994
Distinction	0	1	5	0.65

$$\begin{aligned}\text{Entropy}\,(\text{academics = "fail"}) &= -\frac{2}{3}\log_2\left(\frac{2}{3}\right) - 0 - \frac{1}{3}\log_2\left(\frac{1}{3}\right) \\ &= 0.918\end{aligned} \tag{3.58}$$

$$\begin{aligned}\text{Entropy}\,(\text{academics = "pass"}) &= -\frac{5}{11}\log_2\left(\frac{5}{11}\right) - \frac{6}{11}\log_2\left(\frac{6}{11}\right) - 0 \\ &= 0.994\end{aligned} \tag{3.59}$$

$$\text{Entropy (academics = "distinction")} = 0 - \frac{1}{6}\log_2\left(\frac{1}{6}\right) - \frac{5}{6}\log_2\left(\frac{5}{6}\right) = 0.65 \tag{3.60}$$

Table 3.21: Speaking - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	6	2	2	1.371
Fair	1	5	0	0.65
Fluent	0	0	4	0

$$\text{Entropy (speaking = "hesitant")} = -\frac{6}{10}\log_2\left(\frac{6}{10}\right) - \frac{2}{10}\log_2\left(\frac{2}{10}\right) - \frac{2}{10}\log_2\left(\frac{2}{10}\right) = 1.371 \tag{3.61}$$

$$\text{Entropy (speaking = "fair")} = -\frac{1}{6}\log_2\left(\frac{1}{6}\right) - \frac{5}{6}\log_2\left(\frac{5}{6}\right) - 0 = 0.65 \tag{3.62}$$

$$\text{Entropy (speaking = "fluent")} = 0 - 0 - \frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0 \tag{3.63}$$

Table 3.22: Creative - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Low	4	1	0	0.722
Medium	3	4	3	1.571
High	0	2	3	0.971

$$\text{Entropy (creative = "low")} = -\frac{4}{5}\log_2\left(\frac{4}{5}\right) - \frac{1}{5}\log_2\left(\frac{1}{5}\right) - 0 = 0.722 \tag{3.64}$$

$$\text{Entropy (creative = "medium")} = -\frac{3}{10}\log_2\left(\frac{3}{10}\right) - \frac{4}{10}\log_2\left(\frac{4}{10}\right) - \frac{3}{10}\log_2\left(\frac{3}{10}\right) = 1.571 \tag{3.65}$$

$$\begin{aligned}\text{Entropy (creative = "high")} &= 0 - \frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) \\ &= 0.971\end{aligned} \tag{3.66}$$

Table 3.23: Sports - For Root Node

	Poor	**Average**	**Excellent**	**Entropy**
Bad	7	3	0	0.881
Good	0	3	3	1
Very Good	0	1	3	0.811

$$\begin{aligned}\text{Entropy (sports = "bad")} &= -\frac{7}{10}\log_2\left(\frac{7}{10}\right) - \frac{3}{10}\log_2\left(\frac{3}{10}\right) - 0 \\ &= 0.881\end{aligned} \tag{3.67}$$

$$\begin{aligned}\text{Entropy (sports = "good")} &= 0 - \frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) \\ &= 1\end{aligned} \tag{3.68}$$

$$\begin{aligned}\text{Entropy (sports = "very good")} &= 0 - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{3}{4}\log_2\left(\frac{3}{4}\right) \\ &= 0.811\end{aligned} \tag{3.69}$$

- **Step 3:** Calculate the average information of each independent variable using the formula given below:

$$\text{Avg. Info.} = \Sigma\left(\left(|S_v| / |S|\right) * \text{Entropy}\left(S_v\right)\right) \tag{3.70}$$

where S_v is the set of rows in S for which the feature column A has value v, $|S_v|$ is the number of rows in S_v and likewise $|S|$ is the number of rows in S.

$$\begin{aligned}I\left(\text{academics}\right) = &\left(\left(\frac{2+0+1}{20} * 0.918\right)\right) + \left(\left(\frac{5+6+0}{20} * 0.994\right)\right) \\ &+ \left(\left(\frac{0+1+5}{20} * 0.65\right)\right) \\ &= 0.8784\end{aligned} \tag{3.71}$$

$$
\begin{aligned}
I\left(\text{speaking}\right) = & \left(\left(\frac{6+2+2}{20} * 1.371\right)\right) + \left(\left(\frac{1+5+0}{20} * 0.65\right)\right) \\
& + \left(\left(\frac{0+0+4}{20} * 0\right)\right) \\
& = 0.8805
\end{aligned}
\tag{3.72}
$$

$$
\begin{aligned}
I\left(\text{creative}\right) = & \left(\left(\frac{4+1+0}{20} * 0.722\right)\right) + \left(\left(\frac{3+4+3}{20} * 1.571\right)\right) \\
& + \left(\left(\frac{0+2+3}{20} * 0.971\right)\right) \\
& = 1.20875
\end{aligned}
\tag{3.73}
$$

$$
\begin{aligned}
I\left(\text{sports}\right) = & \left(\left(\frac{7+3+0}{20} * 0.881\right)\right) + \left(\left(\frac{0+3+3}{20} * 1\right)\right) \\
& + \left(\left(\frac{1+3+0}{20} * 0.811\right)\right) \\
& = 0.9027
\end{aligned}
\tag{3.74}
$$

- **Step 4:** Calculate the gain of each independent variable using the formula 3.2

$$
\begin{aligned}
\text{Gain}\left(\text{academics}\right) &= E\left(S\right) - I\left(academics\right) \\
&= 1.581 - 0.8794 = 0.7016
\end{aligned}
\tag{3.75}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{speaking}\right) &= E\left(S\right) - I\left(speaking\right) \\
&= 1.581 - 0.8805 = 0.7005
\end{aligned}
\tag{3.76}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{creative}\right) &= E\left(S\right) - I\left(creative\right) \\
&= 1.581 - 1.20875 = 0.3722
\end{aligned}
\tag{3.77}
$$

$$
\begin{aligned}
\text{Gain}\left(\text{sports}\right) &= E\left(S\right) - I\left(sports\right) \\
&= 1.581 - 0.9027 = 0.6783
\end{aligned}
\tag{3.78}
$$

- **Step 5:** Calculate the SplitInfo of each independent variable using the formula:

$$
\text{SplitInfo}\left(S, A\right) = -\sum_{i} \frac{|S_i|}{|S|} \log\left(\frac{|S_i|}{|S|}\right)
\tag{3.79}
$$

$$\text{SplitInfo}\,(\text{academics}) = -\frac{3}{20}\log_2\left(\frac{3}{20}\right) - \frac{11}{20}\log_2\left(\frac{11}{20}\right) - \frac{6}{20}\log_2\left(\frac{6}{20}\right) \\ = 1.406 \tag{3.80}$$

$$\text{SplitInfo}\,(\text{speaking}) = -\frac{10}{20}\log_2\left(\frac{10}{20}\right) - \frac{6}{20}\log_2\left(\frac{6}{20}\right) - \frac{4}{20}\log_2\left(\frac{4}{20}\right) \\ = 1.485 \tag{3.81}$$

$$\text{SplitInfo}\,(\text{creative}) = -\frac{5}{20}\log_2\left(\frac{5}{20}\right) - \frac{10}{20}\log_2\left(\frac{10}{20}\right) - \frac{5}{20}\log_2\left(\frac{5}{20}\right) \\ = 1.5 \tag{3.82}$$

$$\text{SplitInfo}\,(\text{sports}) = -\frac{10}{20}\log_2\left(\frac{10}{20}\right) - \frac{6}{20}\log_2\left(\frac{6}{20}\right) - \frac{4}{20}\log_2\left(\frac{4}{20}\right) \\ = 1.485 \tag{3.83}$$

- **Step 6:** Calculate the Gain Ratio of each independent variable using the formula 3.4.

$$\text{Gain Ratio}\,(academics) = \frac{0.7016}{1.406} = 0.499 \tag{3.84}$$

$$\text{Gain Ratio}\,(speaking) = \frac{0.7005}{1.485} = 0.471 \tag{3.85}$$

$$\text{Gain Ratio}\,(creative) = \frac{0.3722}{1.5} = 0.248 \tag{3.86}$$

$$\text{Gain Ratio}\,(sports) = \frac{0.6783}{1.485} = 0.456 \tag{3.87}$$

- **Step 7:** Choose the attribute with the highest gain and then repeat steps 1 to 6 for the dataset formed with the chosen attribute.

  Since "Academics" has the highest gain ratio, we choose it as the root node.

Table 3.24: Deciding the Root Node

Attributes	**Gain Ratio**
Academics	**0.499**
Speaking	0.471
Creative	0.248
Sports	0.456

Total number of data points with academics → distinction = 6. Now, repeating all the steps again:

$$\begin{aligned}\text{Entropy}\left(S_{\text{distinction}}\right) &= -\frac{5}{6}\log_2\left(\frac{5}{6}\right) - \frac{1}{6}\log_2\left(\frac{1}{6}\right) - 0 \\ &= 0.65\end{aligned} \tag{3.88}$$

Table 3.25: Speaking - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	0	1	1	1
Fair	0	0	0	0
Fluent	0	0	4	0

$$\text{Entropy}\left(\text{speaking = "hesitant"}\right) = 0 - \frac{1}{2}\log_2\left(\frac{1}{2}\right) - \frac{1}{2}\log_2\left(\frac{1}{2}\right) = 1 \tag{3.89}$$

$$\text{Entropy}\left(\text{speaking = "fair"}\right) = 0 \tag{3.90}$$

$$\text{Entropy}\left(\text{speaking = "fluent"}\right) = -\frac{4}{4}\log_2\left(\frac{4}{4}\right) = 0 \tag{3.91}$$

$$I\left(\text{speaking}\right) = \left(\frac{2}{6} * 1\right) = 0.333 \tag{3.92}$$

$$\begin{aligned}\text{Gain}\left(\text{speaking}\right) &= E\left(S_{distinction}\right) - I\left(speaking\right) \\ &= 0.65 - 0.33 = 0.32\end{aligned} \tag{3.93}$$

$$\text{SplitInfo}\left(\text{speaking}\right) = -\frac{2}{6}\log_2\left(\frac{2}{6}\right) - \frac{4}{6}\log_2\left(\frac{4}{6}\right) = 0.918 \tag{3.94}$$

$$\text{Gain Ratio}\left(\text{speaking}\right) = \frac{0.32}{0.918} = 0.3485 \tag{3.95}$$

Table 3.26: Creative - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Low	0	1	0	0
Medium	0	0	3	0
High	0	0	2	0

$$\text{Entropy}\left(\text{creative = "low"}\right) = 0 \tag{3.96}$$

$$\text{Entropy}\left(\text{creative = "medium"}\right) = 0 \tag{3.97}$$

$$\text{Entropy}\left(\text{creative = "high"}\right) = 0 \tag{3.98}$$

$$I\,(\text{creative}) = 0 \tag{3.99}$$

$$\begin{aligned}\text{Gain}\,(\text{creative}) &= E\,(S_{distinction}) - I\,(creative)\\ &= 0.65 - 0 = 0.65\end{aligned} \tag{3.100}$$

$$\text{SplitInfo}\,(\text{creative}) = -\frac{1}{6}\log_2\left(\frac{1}{6}\right) - \frac{3}{6}\log_2\left(\frac{3}{6}\right) - \frac{2}{6}\log_2\left(\frac{2}{6}\right) = 1.459 \tag{3.101}$$

$$\text{Gain Ratio}\,(\text{creative}) = \frac{0.65}{1.459} = 0.4455 \tag{3.102}$$

Table 3.27: Sports - For Academics→"distinction"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	0	0	0	0
Good	0	1	3	0.811
Very Good	0	0	2	0

$$\text{Entropy}\,(\text{sports = "bad"}) = 0 \tag{3.103}$$

$$\text{Entropy}\,(\text{sports = "good"}) = 0 - \frac{1}{4}\log_2\left(\frac{1}{4}\right) - \frac{1}{3}\log_2\left(\frac{1}{3}\right) = 0.811 \tag{3.104}$$

$$\text{Entropy}\,(\text{sports = "very good"}) = 0 \tag{3.105}$$

$$I\,(\text{sports}) = \left(\frac{4}{6} * 0.811\right) = 0.540 \tag{3.106}$$

$$\begin{aligned}\text{Gain}\,(\text{sports}) &= E\,(S_{distinction}) - I\,(sports)\\ &= 0.65 - 0.540 = 0.11\end{aligned} \tag{3.107}$$

$$\text{SplitInfo}\,(\text{sports}) = 1.028 \tag{3.108}$$

$$\text{Gain Ratio}\,(\text{sports}) = 0.1064 \tag{3.109}$$

Next node will be "Creative":

Now, for Academics→"Pass", total number of data points = 11.

$$\text{Entropy}\,(S_{\text{pass}}) = 0.994 \tag{3.110}$$

Table 3.28: Deciding the Node after Academics→"Distinction"

Attributes	**Gain Ratio**
Speaking	0.3485
Creative	**0.4455**
Sports	0.1064

Table 3.29: Speaking - For Academics→"pass"

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	4	1	0	0.722
Fair	1	5	0	0.65
Fluent	0	0	0	0

$$I\left(\text{speaking}\right) = \left(\frac{5}{11} * 0.722\right) + \left(\frac{6}{11} * 0.65\right) = 0.6827 \tag{3.111}$$

$$\text{Gain}\left(\text{speaking}\right) = 0.3113 \tag{3.112}$$

$$\text{SplitInfo}\left(\text{speaking}\right) = 0.994 \tag{3.113}$$

$$\text{Gain Ratio}\left(\text{speaking}\right) = \frac{0.3113}{0.994} = 0.313 \tag{3.114}$$

Table 3.30: Creative - For Academics→"pass"

	Poor	**Average**	**Excellent**	**Entropy**
Low	3	0	0	0
Medium	2	4	0	0.918
High	0	2	0	0

$$I\left(\text{creative}\right) = \left(\frac{6}{11} * 0.918\right) = 0.5 \tag{3.115}$$

$$\text{Gain}\left(\text{creative}\right) = 0.494 \tag{3.116}$$

$$\text{SplitInfo}\left(\text{creative}\right) = 1.435 \tag{3.117}$$

$$\text{Gain Ratio}\left(\text{creative}\right) = \frac{0.494}{1.435} = 0.344 \tag{3.118}$$

$$I\left(\text{sports}\right) = \left(\frac{8}{11} * 0.954\right) = 0.6938 \tag{3.119}$$

Table 3.31: Sports - For Academics→"pass"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	5	3	0	0.954
Good	0	2	0	0
Very Good	0	1	0	0

$$\text{Gain}\,(\text{sports}) = 0.3 \tag{3.120}$$

$$\text{SplitInfo}\,(\text{sports}) = 1.096 \tag{3.121}$$

$$\text{Gain Ratio}\,(\text{sports}) = \frac{0.30}{1.096} = 0.273 \tag{3.122}$$

Table 3.32: Deciding the Node after Academics→"Pass"

Attributes	**Gain**
Speaking	0.313
Creative	**0.344**
Sports	0.273

Now, for Academics→"Fail", total number of data points = 3.

$$\text{Entropy}\,(S_{\text{pass}}) = 0.918 \tag{3.123}$$

Table 3.33: Speaking - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Hesitant	2	0	1	0.918
Fair	0	0	0	0
Fluent	0	0	0	0

$$I\,(\text{speaking}) = \left(\frac{3}{3} * 0.918\right) = 0.918 \tag{3.124}$$

$$\text{Gain}\,(\text{speaking}) = 0 \tag{3.125}$$

$$\text{SplitInfo}\,(\text{speaking}) = 0 \tag{3.126}$$

$$\text{Gain Ratio}\,(\text{speaking}) = 0 \tag{3.127}$$

$$I\,(\text{creative}) = 0 \tag{3.128}$$

Table 3.34: Creative - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Low	1	0	0	0
Medium	1	0	0	0
High	0	0	1	0

$$\text{Gain (creative)} = 0.918 \tag{3.129}$$

$$\text{SplitInfo (creative)} = 1.585 \tag{3.130}$$

$$\text{Gain Ratio (creative)} = 0.579 \tag{3.131}$$

Table 3.35: Sports - For Academics→"Fail"

	Poor	**Average**	**Excellent**	**Entropy**
Bad	2	0	0	0
Good	0	0	0	0
Very Good	0	0	1	0

$$I\,(\text{sports}) = 0 \tag{3.132}$$

$$\text{Gain (sports)} = 0.918 \tag{3.133}$$

$$\text{SplitInfo (sports)} = 0.918 \tag{3.134}$$

$$\text{Gain Ratio (sports)} = \frac{0.918}{0.918} = 1 \tag{3.135}$$

Table 3.36: Deciding the Node after Academics→"Fail"

Attributes	**Gain**
Speaking	0
Creative	0.579
Sports	**1**

Similarly, we continue for all the nodes, we finally get figure 3.3.

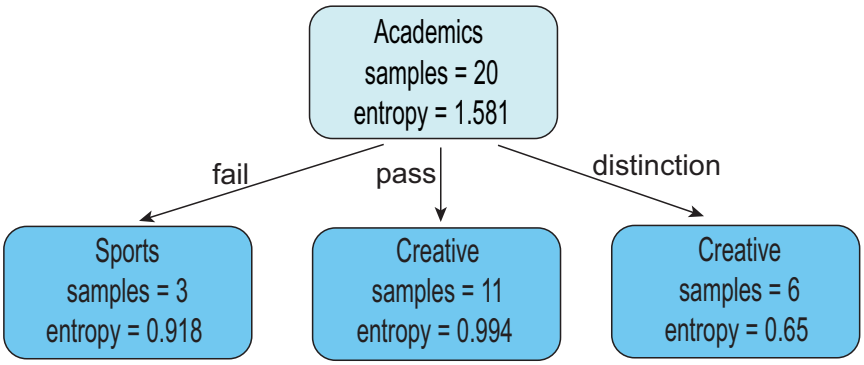


Figure 3.8: Node under Academics→"Fail"

Chapter 4

BAYESIAN APPROACH

4.1 Introduction to Bayesian Approach

4.1.1 Bayes' Theorem

Bayes' theorem, also known as Bayes' rule (named for Reverend Thomas Bayes), is a simple yet principled mathematical formula in the field of probability theory and statistics. This theorem is widely used in Bayesian approaches to statistics, epistemology and inductive logic to calculate conditional probabilities, i.e., the probability of an event based on the prior knowledge of other related conditions [23].

$$P(A \mid B) = \frac{P(B \mid A) \cdot P(A)}{P(B)} \tag{4.1}$$

where,

A, B $\rightarrow$ events;

P(A $\mid$ B) $\rightarrow$ probability of A given B is true;

P(B $\mid$ A) $\rightarrow$ probability of B given A is true;

P(A), P(B) $\rightarrow$ the independent probabilities of A and B

4.1.2 Bayes' Theorem in Data Science

A powerful tool in the field of probability and statistics, Bayes' Theorem is widely used in the field of data science. The wide adoption of Bayes' Theorem in the field of data science is attributed to the fact that several data science problems are probabilistic in nature. Data science algorithms based on Bayes' Theorem includes Bayesian Belief Network, Naive Bayes, among others [24].

4.2 Solved Numerical

4.2.1 Question

Assume you have won ₹1000 in a game show. Now, the game show host offers you two wallets. In wallet A, there are twenty ₹100 bills (making ₹2000), while in wallet B, there are five ₹10 bills and five ₹100 bills (making ₹550). In total, the host gives you three options: 1) Keep the ₹1000 and quit 2) For a cost of ₹1000, choose one of the two wallets (meaning that essentially you would win either ₹2000 or ₹550) or 3) For a cost of ₹200, pull one of the bills out of any of the wallet before choosing whether or not to take the contents (which will be either ₹2000 or ₹550) at the additional cost of ₹1000. (a) Structure the decision tree for determining whether you should keep your ₹1000, take the wallet without pulling a bill (at the cost of ₹1000), or look at a bill before deciding whether or not you should buy the wallet. Calculate the appropriate consequences and probabilities for the tree. (b) What is the strategy that maximizes the expected monetary value?

4.2.2 Solution

In the given situation, the player has three options:

1. Either he/she can take the initial prize without any risk, or,
2. he/she can take one of the two wallets randomly, or,
3. he/she can choose to look at a bill before deciding whether to take that wallet or take the initial prize minus the additional cost, which is ₹200 in this case.

To maximize the expected monetary benefit, we will use the Bayes' theorem to get probability inputs and calculate expected gain in all three scenarios.

$$P(\text{wallet}_\text{A}) = P(\text{wallet}_\text{B}) = 0.5 \tag{4.2}$$

The likelihood ratios of a ₹10 and ₹100 bill being pulled from wallet A is given by the following two equations:

$$P(₹10 \mid \text{wallet}_\text{A}) = \frac{0}{20} = 0 \tag{4.3}$$

$$P(₹100 \mid \text{wallet}_\text{A}) = \frac{20}{20} = 1 \tag{4.4}$$

Similarly, for wallet B:

$$P(₹10 \mid \text{wallet}_\text{B}) = \frac{5}{10} = 0.5 \tag{4.5}$$

$$P(₹100 \mid \text{wallet}_\text{B}) = \frac{5}{10} = 0.5 \tag{4.6}$$

Total probabilities of ₹10 bill and ₹100 being pulled are calculated as:

$$\begin{aligned} P(₹10) &= P(₹10 \mid \text{wallet}_\text{A}) * P(\text{wallet}_\text{A}) + P(₹10 \mid \text{wallet}_\text{B}) * P(\text{wallet}_\text{B}) \\ &= 0 + 0.5 * 0.5 = 0.25 \end{aligned} \tag{4.7}$$

$$\begin{aligned} P(₹100) &= P(₹100 \mid \text{wallet}_\text{A}) * P(\text{wallet}_\text{A}) + P(₹100 \mid \text{wallet}_\text{B}) * P(\text{wallet}_\text{B}) \\ &= 1 * 0.5 + 0.5 * 0.5 = 0.75 \end{aligned} \tag{4.8}$$

Now, calculating conditional probabilities using Bayes' theorem - For wallet A:

$$P(\text{wallet}_\text{A} \mid ₹10) = \frac{P(₹10 \mid \text{wallet}_\text{A})}{P(₹10)} * P(\text{wallet}_\text{A}) = \frac{0}{0.25} * 0.5 = 0 \tag{4.9}$$

$$P(\text{wallet}_\text{A} \mid ₹100) = \frac{P(₹100 \mid \text{wallet}_\text{A})}{P(₹100)} * P(\text{wallet}_\text{A}) = \frac{1}{0.75} * 0.5 = 0.6666 \tag{4.10}$$

For wallet B:

$$P(\text{wallet}_\text{B} \mid ₹10) = \frac{P(₹10 \mid \text{wallet}_\text{B})}{P(₹10)} * P(\text{wallet}_\text{B}) = \frac{0.5}{0.25} * 0.5 = 1.0 \tag{4.11}$$

$$P(\text{wallet}_\text{B} \mid ₹100) = \frac{P(₹100 \mid \text{wallet}_\text{B})}{P(₹100)} * P(\text{wallet}_\text{B}) = \frac{0.5}{0.75} * 0.5 = 0.3333 \tag{4.12}$$

Now that we have all the conditional probabilities, we will calculate the expected gain in all three scenarios:

Scenario 1: Keeping the Money

$$EG_{s1} = ₹1000 \tag{4.13}$$

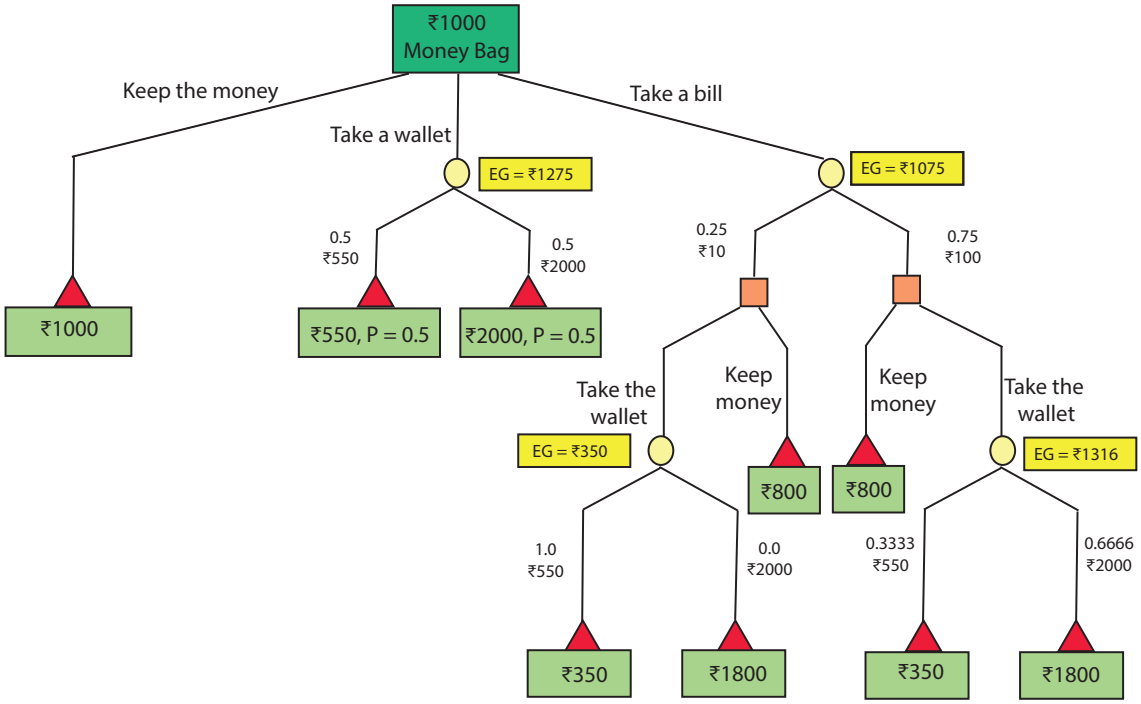


Figure 4.1: Part (a) of the Solution: Decision Tree

Scenario 2: Taking a Wallet Randomly

$$\begin{aligned} EG_{s2} &= ₹2000 * P(\text{wallet}_\text{A}) + ₹550 * P(\text{wallet}_\text{B}) \\ &= 2000 * 0.5 + 550 * 0.5 = ₹1275 \end{aligned} \tag{4.14}$$

Scenario 3: Taking a Bill

$$EG_{s3} = EG_{s3.1} * P(₹10) * + EG_{s3.2} * P(₹100) * \tag{4.15}$$

where:

$$\begin{aligned} EG_{s3.1} &= ₹(2000 - 200) * P(\text{wallet}_\text{A} \mid ₹10) + ₹(550 - 200) * P(\text{wallet}_\text{B} \mid ₹10) \\ &= 1800 * 0 + 350 * 1 = ₹350 \end{aligned} \tag{4.16}$$

$$\begin{aligned} EG_{s3.2} &= ₹(2000 - 200) * P(\text{wallet}_\text{A} \mid ₹100) + ₹(550 - 200) * P(\text{wallet}_\text{B} \mid ₹100) \\ &= 1800 * 0.6666 + 350 * 0.3333 \approx ₹1316 \end{aligned} \tag{4.17}$$

Using eqns 4.16 & 4.17 in eqn 4.15, we get:

$$\begin{aligned} EG_{s3} &= ₹350 * P(₹10) + ₹1316 * P(₹100) = ₹350 * 0.25 + ₹1316 * 0.75 \\ &= ₹87.5 + ₹987 \approx ₹1075 \end{aligned} \tag{4.18}$$

(b) The maximum expected gain is in **scenario 2**. Therefore, the player should be advised to **take a wallet randomly**.

4.3 Python Implementation of Bayesian Decision Tree

See Appendix B.5.

Chapter 5

CONCLUSION

5.1 Conclusion

In this project work, we examined two main topics in the field of data science - namely SVM and Decision Trees. We studied the various types of SVM algorithms, including linear and non-linear SVM. In addition to implementing SVM, we also solved a regression problem using Support Vector Regression (SVR). Further, we studied the basics of the decision tree model, its various types (Classification and Regression Tree (CART), Iterative Dichotomiser 3 (ID3), C4.5), and the terminology involved (entropy, gain ratio, information gain, Gini index). For both SVM and Decision Tree algorithm, we also studied their advantages, disadvantages and application areas. To gain in-depth knowledge of the working of decision trees as well as to understand the difference between ID3 and C4.5 algorithms, we implemented both of them using the python programming language, as well as manually. Lastly, we studied the Bayesian approach to classification. We solved a popular question using Bayes' theorem, both manually and using python.

Appendix A

PROJECT SPECIFICATIONS

A.1 Hardware Specifications

A.1.1 Recommended Configurations

- **CPU Speed:** 1.4 GHz or higher recommended (per core)
- **Processor:** Intel Core i5-8100 (8th Gen) Dual Core or above
- **Memory/ RAM:** 8GB or higher
- **Storage:** 300 GB Hard Disk Drive (HDD)/ 128 GB Solid State Drive (SSD) or above

A.1.2 Minimum Configurations

- **CPU Speed:** 1.2 GHz (per core)
- **Processor:** Intel Core i3-6100 (6th Gen) Dual Core
- **Memory/ RAM:** 4GB
- **Storage:** 100 GB HDD/ 64 GB SSD

A.2 Software Specifications & Requirements

- **Operating Systems Used:** Microsoft Windows 10 Version 10.0.x and Ubuntu
- **Programming Language Used:** Python 3.7.x 64 bit
- **IDEs/ Environments Used:**
  Spyder 3.3.6
  VS Code 1.44.0
  Jupyter Notebook 6.0.3

A.2.1 Project Compatibility

- **Operating Systems:**
  Microsoft Windows 7 or above
  Mac OS El Capitan or above
  Linux Flavours - Debian/Ubuntu, Fedora, Mandriva, Slackware, ArchLinux, Flatpak, openSUSE, RHEL
- **Python Version:** Python 3.x.x
- **IDEs:** Compatible with all Python IDEs provided that all the required packages and modules are installed.
- **Recommended IDEs:** Spdyer 3.3.x and VS Code 1.35.1 or above

How to install the packages in Python?

- In case of standalone python IDE:
  - Using pip3 installer in terminal.
- In case of anaconda environment:
  - Using conda install or pip3 installer in anaconda prompt.

Appendix B

PYTHON CODES

B.1 SVM Classification Code

```
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from matplotlib.colors import ListedColormap
import matplotlib.pyplot as plt
import warnings

data_df = pd.read_csv('./dataset.csv')

x_df = data_df[:][['Len1', 'Len2']]
y_df = data_df['Output']

train_x, test_x, train_y, test_y = train_test_split(x_df, y_df,
   test_size=.3, random_state=0)

standardscaler = StandardScaler()

standardscaler.fit(train_x)

train_x_std = standardscaler.transform(train_x)
test_x_std = standardscaler.transform(test_x)

svm_model = SVC(kernel='rbf', random_state=0, gamma=.10, C=1.0)
```

```
svm_model.fit(train_x_std, train_y)

print('Accuracy of model on training data is {:.2f}'.format(
   svm_model.score(train_x_std, train_y)))

print('Accuracy of model on test data is {:.2f}'.format(svm_model.
   score(test_x_std, test_y)))

def tuple_v(v):
   return tuple(map(int, (v.split("."))))

def plot_func(x_df, y_df, model, idx_test=None, res=0.02):

   mark = ('s', 'x', 'o', '^', 'v')
   cols = ('#edf285', '#ec5858', '#fd8c04', '#93abd3', '#f9f7cf')
   map_c = ListedColormap(cols[:len(np.unique(y_df))])

   x_min_1, x_max_1 = x_df[:, 0].min() - 1, x_df[:, 0].max() + 1
   x_min_2, x_max_2 = x_df[:, 1].min() - 1, x_df[:, 1].max() + 1
   x1_np, x2_np = np.meshgrid(np.arange(x_min_1, x_max_1, res),
                    np.arange(x_min_2, x_max_2, res))
   z_np = model.predict(np.array([x1_np.ravel(), x2_np.ravel()]).T)
   z_np = z_np.reshape(x1_np.shape)
   plt.contourf(x1_np, x2_np, z_np, alpha=0.4, cmap=map_c)
   plt.xlim(x1_np.min(), x1_np.max())
   plt.ylim(x2_np.min(), x2_np.max())

   for idx, cl in enumerate(np.unique(y_df)):
      plt.scatter(x=x_df[y_df == cl, 0], y=x_df[y_df == cl, 1],
         alpha=0.8, c=map_c(idx), marker=mark[idx], label=cl)
   plt.show()

plot_func(test_x_std, test_y, svm_model)
```

B.2 SVM Regression Code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from pandas import DataFrame
import datetime

from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import KFold
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV

train_df = pd.read_csv('./train.csv')
test_df = pd.read_csv('./test.csv')

output = train_df['output']
del train_df['output']

all_df = pd.concat([train_df,test_df],axis=0)
all_df['Start'] = pd.to_datetime(all_df["Start"])
all_df['Year'] = all_df['Start'].apply(lambda x:x.year)
all_df['Month'] = all_df['Start'].apply(lambda x:x.month)
all_df['Day'] = all_df['Start'].apply(lambda x:x.day)

labelencoder = LabelEncoder()
all_df['Place'] = labelencoder.fit_transform(all_df['Place'])
all_df['Group'] = all_df['Group'].map({'Other':0,'Big Cities':1})
all_df["Category"] = all_df["Category"].map({"FC":0, "IL":1, "DT":2,
    "MB":3})
```

```
train_df = all_df.iloc[:train_df.shape[0]]
test_df = all_df.iloc[train_df.shape[0]:]

train_cols_df = [col for col in train_df.columns if col not in ['Id
   ', 'Start']]

standardscaler = StandardScaler()
minmaxscaler = MinMaxScaler()

train_sc_df = standardscaler.fit_transform(train_df[train_cols_df])
train_scms_df = minmaxscaler.fit_transform(train_sc_df)

def gen_cv():
   m_train = np.floor(len(output)*0.75).astype(int)
   train_idx = np.arange(m_train)
   test_idx = np.arange(m_train, len(output))
   yield (train_idx, test_idx)

cnt_param = 20
param = {"C":np.logspace(0,1,cnt_param), "epsilon":np.logspace(-1,1,
   cnt_param)}
gridsearch = GridSearchCV(SVR(kernel="linear"), param, cv=gen_cv(),
scoring="r2", return_train_score=True)
gridsearch.fit(train_scms_df, output)
print('The best parameter = ',gridsearch.best_params_)
print('accuracy = ',gridsearch.best_score_)

regr = SVR(kernel="linear", C=gridsearch.best_params_["C"], epsilon=
   gridsearch.best_params_["epsilon"])

cnt_split = 5
kfold = KFold(n_splits=cnt_split,shuffle=True,random_state=0)

svr_models = []
```

```
for train_index,test_index in kfold.split(train_scms_df):
   train_x = train_df.iloc[train_index][train_cols_df]
   train_y = output.iloc[train_index]
   test_x = train_df.iloc[test_index][train_cols_df]
   test_y = output.iloc[test_index]
   svr_model = regr.fit(train_x, train_y)
   svr_models.append(svr_model)
   pred = regr.predict(test_x)
   plt.scatter(train_x['P11'], train_y)
   plt.plot(test_x['P11'], pred, color='red')
   plt.xlabel('P11')
   plt.ylabel('Output')
   plt.show()
```

B.3 ID3 Code

```
import re
import numpy
import math
import random

class Node:

   def __init__(self, data):
      self.left = None
      self.right = None
      self.data = data
      self.threshold_indices = -1
      self.threshold = -1
      self.entropy=-1
      self.leaf = True
      self.pure = True
      self.label = -1
```

```
    if len(data) > 1:
      label = data[0][1]
      for i in range(1, len(data)):
        if label != data[i][1]:
          self.pure = False
    if self.pure:
      self.label = data[0][1]

def setThresholdIndices(self, index):
  self.threshold_indices = index

def setThreshold(self, val,ent):
  self.threshold=val
  self.entropy=ent


def setEntropy(self, val):
  self.entropy = val

def setLeft(self, elem):
  self.leaf = False
  self.left = elem

def setRight(self, elem):
  self.leaf = False
  self.right = elem

def getThreshold(self):
  return self.threshold

def getentropy(self):
  return self.entropy

def getThresholdIndices(self):
```

```
      return self.threshold_indices

    def getRight(self):
      return self.right

    def getLeft(self):
      return self.left

    def getData(self):
      return self.data

    def getLabel(self):
      return self.label

    def isLeaf(self):
      return self.leaf

    def isPure(self):
      return self.pure

    def getLabelOrThreshold(self):
       if (not self.pure):
          return "threshold: " + str(self.threshold) +" entropy " +
             str(self.entropy)
       if (self.pure):
          return "label: " + str(self.label)

def load_data(filename):
    data_matrix = []
    f = open(filename, "r")
    for line in f:
      tokens=line.split()
      features=[]
      for i in range(0, len(tokens)-1):
```

```
        features.append(float(tokens[i]))
        data=(features, int(tokens[len(tokens)-1]))
        data_matrix.append(data)
    return data_matrix

# calculates the entropy, given that there are only 3 valid
    labelings 1,2,3
def calc_entropy(data):
  count = [0,0,0,0]
  total = float(0)
  # really only need indices 1,2,3 as those are the only labels
  for (features, label) in data:
    count[label] = count[label] + 1
    total = total + 1
  entropy = float(0)
  for c in count:
    if c == 0:
      continue
    prob = c / total
    entropy = entropy - prob * math.log(prob)
  return entropy

# split the data into a left(true branch) and right(false branch)
    given a dataset, threshold, and feature index
def split(dataset, threshold, feature_index):
  left = []
  right = []
  for datapoint in dataset:
    if datapoint[0][feature_index] <= threshold:
      left.append(datapoint)
    else:
      right.append(datapoint)
  return (left,right)
```

```
#whats the lowest entropy and threshold I can get given a dataset
   and a specific feature index
def calc_lowest_entropy(dataset, feature_index):
  sort = sorted(dataset, key=lambda tup: tup[0][feature_index])
  best_entropy = float('inf')
  best_thres = float('inf')
  curr_entropy = float('inf')
  curr_thres = float('inf')

  for i in range(0, len(dataset)):
    if curr_thres == dataset[i][0][feature_index]:
      continue
    curr_thres = dataset[i][0][feature_index]
    (left,right) = split(dataset, curr_thres, feature_index)
    curr_entropy = calc_entropy(left) * float(len(left))/float(len(
       dataset)) + calc_entropy(right) * float(len(right))/float(len(
       dataset))
    if curr_entropy < best_entropy:
      best_entropy = curr_entropy
      best_thres = curr_thres
  return (best_entropy, best_thres)

# what the threshold, and feature index to split by given a dataset
def calc_threshold(dataset):
  best_feature_index = -1
  best_entropy = float('inf')
  best_threshold = float('inf')

  for i in range(0, len(dataset[0][0])):
    (entropy, thres) = calc_lowest_entropy(dataset, i)
    if entropy < best_entropy:
      best_entropy = entropy
      best_feature_index = i
      best_threshold = thres
```

```
  return (best_entropy, best_threshold, best_feature_index)

def find_impure_leaf(node):
  if node is None:
    return None
  if not(node.isPure()) and node.isLeaf():
    return node

  lefty = find_impure_leaf(node.getLeft())
  if not(lefty is None):
    return lefty

  righty = find_impure_leaf(node.getRight())
  if not(righty is None):
    return righty

  return None

def ID3(root):

  curr_node = find_impure_leaf(root)
  while not(curr_node is None):
    (entropy, threshold, feature_index) = calc_threshold(curr_node.
       getData())
    (left, right) = split(curr_node.getData(), threshold,
       feature_index)

    curr_node.setThreshold(threshold,entropy)
    curr_node.setThresholdIndices(feature_index)
#   curr_node.setEntropy(entropy)

    left_node = Node(left)
    right_node = Node(right)
```

```
    curr_node.setLeft(left_node)
    curr_node.setRight(right_node)
    curr_node = find_impure_leaf(root)

def calc_error(dataset, root):
  errors = 0
  num_samples = len(dataset)
  for (features, label) in dataset:
    curr_node = root
    while not(curr_node.isPure()):
      threshold = curr_node.getThreshold()
      feature_index = curr_node.getThresholdIndices()
      if features[feature_index] <= threshold:
        curr_node = curr_node.getLeft()
      else:
        curr_node = curr_node.getRight()
    if not(label == curr_node.getLabel()):
      errors = errors + 1
  return float(errors) / float(num_samples)

def print_tree(root):
      thislevel = [root]
      while thislevel:
        nextlevel = list()
        for n in thislevel:
          print (n.getLabelOrThreshold(), "training points: " + str(len
              (n.getData())))
         # print (n.getentropy())
          if n.getLeft():
             nextlevel.append(n.getLeft())
          if n.getRight():
             nextlevel.append(n.getRight())
        print ()
        thislevel = nextlevel
```

```
def main():
   training_set = load_data("hw3train_edit.txt")
   test_set = load_data("hw3test_edit.txt")
   sort_test = sorted(training_set, key=lambda tup: tup[0][2])
   root = Node(training_set)
   ID3(root)
   print (calc_error(training_set, root))
   print (calc_error(test_set, root))
   print_tree(root)

if __name__ == '__main__':
   main()
```

B.4 C4.5 Code

```
import math
class C45:

  """Creates a decision tree with C4.5 algorithm"""
  def __init__(self, pathToData,pathToNames):
    self.filePathToData = pathToData
    self.filePathToNames = pathToNames
    self.data = []
    self.classes = []
    self.numAttributes = -1
    self.attrValues = {}
    self.attributes = []
    self.tree = None

  def fetchData(self):
    with open(self.filePathToNames, "r") as file:
      classes = file.readline()
```

```
    self.classes = [x.strip() for x in classes.split(",")]
    #add attributes
    for line in file:
      [attribute, values] = [x.strip() for x in line.split(":")]
      values = [x.strip() for x in values.split(",")]
      self.attrValues[attribute] = values
  self.numAttributes = len(self.attrValues.keys())
  self.attributes = list(self.attrValues.keys())
  with open(self.filePathToData, "r") as file:
    for line in file:
      row = [x.strip() for x in line.split(",")]
      if row != [] or row != [""]:
        self.data.append(row)

def preprocessData(self):
  for index,row in enumerate(self.data):
    for attr_index in range(self.numAttributes):
      if(not self.isAttrDiscrete(self.attributes[attr_index])):
        self.data[index][attr_index] = float(self.data[index][
            attr_index])

def printTree(self):
  self.printNode(self.tree)

def printNode(self, node, indent=""):
  if not node.isLeaf:
    if node.threshold is None:
      #discrete
      for index,child in enumerate(node.children):
        if child.isLeaf:
          print(indent + node.label + " = " + attributes[index] + " :
              " + child.label)
        else:
          print(indent + node.label + " = " + attributes[index] + " :
```

```
            ")
          self.printNode(child, indent + " ")
      else:
        #numerical
        leftChild = node.children[0]
        rightChild = node.children[1]
        if leftChild.isLeaf:
          #print(node.entropy)
          print(indent + node.label + " <= " + str(node.threshold) + "
              : " + leftChild.label)
        else:
          print(indent + node.label + " <= " + str(node.threshold)+" :
              ")
          self.printNode(leftChild, indent + " ")

        if rightChild.isLeaf:
          print(indent + node.label + " > " + str(node.threshold) + " :
               " + rightChild.label)
        else:
          print(indent + node.label + " > " + str(node.threshold) + " :
               ")
          self.printNode(rightChild , indent + " ")


def generateTree(self):
  self.tree = self.recursiveGenerateTree(self.data, self.attributes
      )

def recursiveGenerateTree(self, curData, curAttributes):
  allSame = self.allSameClass(curData)

  if len(curData) == 0:
    #Fail
    return Node(True, "Fail", None)
```

```
    elif allSame is not False:
      #return a node with that class
      return Node(True, allSame, None)
    elif len(curAttributes) == 0:
      #return a node with the majority class
      majClass = self.getMajClass(curData)
      return Node(True, majClass, None)
    else:
      (best,best_threshold,splitted) = self.splitAttribute(curData,
          curAttributes)
      remainingAttributes = curAttributes[:]
      remainingAttributes.remove(best)
      node = Node(False, best, best_threshold)
      node.children = [self.recursiveGenerateTree(subset,
          remainingAttributes) for subset in splitted]
      return node

  def getMajClass(self, curData):
    freq = [0]*len(self.classes)
    for row in curData:
      index = self.classes.index(row[-1])
      freq[index] += 1
    maxInd = freq.index(max(freq))
    return self.classes[maxInd]

  def allSameClass(self, data):
    for row in data:
      if row[-1] != data[0][-1]:
        return False
    return data[0][-1]

  def isAttrDiscrete(self, attribute):
    if attribute not in self.attributes:
      raise ValueError("Attribute not listed")
```

```
  elif len(self.attrValues[attribute]) == 1 and self.attrValues[
     attribute][0] == "continuous":
    return False
  else:
    return True

def splitAttribute(self, curData, curAttributes):
  splitted = []
  maxEnt = -1*float("inf")
  best_attribute = -1
  #None for discrete attributes, threshold value for continuous
     attributes
  best_threshold = None
  for attribute in curAttributes:
    indexOfAttribute = self.attributes.index(attribute)
    if self.isAttrDiscrete(attribute):
      #split curData into n-subsets, where n is the number of
      #different values of attribute i. Choose the attribute with
      #the max gain
      valuesForAttribute = self.attrValues[attribute]
      subsets = [[] for a in valuesForAttribute]
      for row in curData:
        for index in range(len(valuesForAttribute)):
          if row[i] == valuesForAttribute[index]:
            subsets[index].append(row)
            break
      e = gain(curData, subsets)
      if e > maxEnt:
        maxEnt = e
        splitted = subsets
        best_attribute = attribute
        best_threshold = None
    else:
      #sort the data according to the column.Then try all
```

```
      #possible adjacent pairs. Choose the one that
      #yields maximum gain
      curData.sort(key = lambda x: x[indexOfAttribute])
      for j in range(0, len(curData) - 1):
       if curData[j][indexOfAttribute] != curData[j+1][
          indexOfAttribute]:
        threshold = (curData[j][indexOfAttribute] + curData[j+1][
          indexOfAttribute]) / 2
        less = []
        greater = []
        for row in curData:
         if(row[indexOfAttribute] > threshold):
          greater.append(row)
         else:
          less.append(row)
        e = self.gain(curData, [less, greater])
        if e >= maxEnt:
         splitted = [less, greater]
         maxEnt = e
         best_attribute = attribute
         best_threshold = threshold
  return (best_attribute,best_threshold,splitted)

def gain(self,unionSet, subsets):
  #input : data and disjoint subsets of it
  #output : information gain
  S = len(unionSet)
  #calculate impurity before split
  impurityBeforeSplit = self.entropy(unionSet)
  #calculate impurity after split
  weights = [len(subset)/S for subset in subsets]
  impurityAfterSplit = 0
  for i in range(len(subsets)):
    impurityAfterSplit += weights[i]*self.entropy(subsets[i])
```

```
    #calculate total gain
    totalGain = impurityBeforeSplit - impurityAfterSplit
    return totalGain

  def entropy(self, dataSet):
    S = len(dataSet)
    if S == 0:
      return 0
    num_classes = [0 for i in self.classes]
    for row in dataSet:
      classIndex = list(self.classes).index(row[-1])
      num_classes[classIndex] += 1
    num_classes = [x/S for x in num_classes]
    ent = 0
    for num in num_classes:
      ent += num*self.log(num)
#   self.entropy=ent*-1
    return ent*-1

  def log(self, x):
    if x == 0:
      return 0
    else:
      return math.log(x,2)

class Node:
  def __init__(self,isLeaf, label, threshold):
    self.label = label
    self.threshold = threshold
    self.isLeaf = isLeaf
    self.children = []

import pdb
```

```
c1 = C45("data_1.data", "data_1.names")
c1.fetchData()
c1.preprocessData()
c1.generateTree()
c1.printTree()
```

B.5 Bayesian Decision Tree Code

```
# Money Bag Problem

# Defining global variables
initial_prize = int(input("Enter the initial money bag amount "))
wallet1_100 = int(input("Enter the number of 100 rupee bills in
   wallet 1 "))
wallet1_10 = int(input("Enter the number of 10 rupee bills in wallet
    1 "))
wallet2_100 = int(input("Enter the number of 100 rupee bills in
   wallet 2 "))
wallet2_10 = int(input("Enter the number of 10 rupee bills in wallet
    2 "))
cost_of_bill_selection = int(input("Enter the cost incurred for
   selecting a bill "))

# We keep the cost of selecting a wallet equal to the initial prize
   so that
# final amount won by the player is equal to the contents of the
   wallet
cost_of_wallet_selection = initial_prize;

def get_max_scenario(a, b, c):
  list = [a, b, c]
  if max(list) == a:
     return "keep the initial prize."
```

```
  elif max(list) == b:
     return "take a wallet randomly."
  else:
     return "take a bill before deciding wether or not to take a
        wallet."

def calc_gain(prob1, amount1, prob2, amount2):
   return ((prob1*amount1) + (prob2*amount2))

def calc_conditional(prob_bill, lr, prob_wallet):
   return ((lr/prob_bill)*prob_wallet)

if __name__=="__main__":
   # defining the probabilities of selecting both the wallets (equal
       probability)
   p_wallet1 = p_wallet2 = 0.5
   amount_wallet1 = wallet1_100*100 + wallet1_10*10
   amount_wallet2 = wallet2_100*100 + wallet2_10*10
   take_a_wallet = calc_gain(p_wallet1, amount_wallet1 +
      initial_prize - cost_of_wallet_selection,
                     p_wallet2, amount_wallet2 + initial_prize -
                        cost_of_wallet_selection)
   # total no. of bills in wallet 1 and wallet 2
   total_wallet1 = wallet1_100 + wallet1_10
   total_wallet2 = wallet2_100 + wallet2_10

   # calculating likelihood ratios
   lr_wallet1_10 = wallet1_10/total_wallet1
   lr_wallet1_100 = wallet1_100/total_wallet1
   lr_wallet2_10 = wallet2_10/total_wallet2
   lr_wallet2_100 = wallet2_100/total_wallet2

   prob_bill_10 = p_wallet1*lr_wallet1_10 + p_wallet2*lr_wallet2_100
   prob_bill_100 = p_wallet1*lr_wallet1_100 + p_wallet2*
```

```
    lr_wallet2_100
keep_money_after_bill_selection = initial_prize -
    cost_of_bill_selection

# using bayes theorem to calculate conditional probabilities
p_wallet1_given_100 = calc_conditional(prob_bill_100,
    lr_wallet1_100, p_wallet1 )
p_wallet1_given_10 = calc_conditional(prob_bill_10, lr_wallet1_10
    , p_wallet1 )
p_wallet2_given_100 = calc_conditional(prob_bill_100,
    lr_wallet2_100, p_wallet2 )
p_wallet2_given_10 = calc_conditional(prob_bill_10, lr_wallet2_10
    , p_wallet2 )

take_selected_wallet_if_bill_100 = calc_gain(p_wallet1_given_100,
     amount_wallet1
                                    - cost_of_bill_selection,
                                       p_wallet2_given_100,
                                    amount_wallet2 -
                                       cost_of_bill_selection)

take_selected_wallet_if_bill_10 = calc_gain(p_wallet1_given_10,
    amount_wallet1 + initial_prize
                                    - cost_of_bill_selection,
                                       p_wallet2_given_10,
                                    amount_wallet2 -
                                       cost_of_bill_selection)
choose_a_bill = calc_gain(prob_bill_10,
    take_selected_wallet_if_bill_10,
                                    prob_bill_100,
                                       take_selected_wallet_if_bill_100
                                       )

print("Scenario 1: You keep the initial prize. Your expected gain
```

```
    is:",
     initial_prize)
print("Scenario 2: You take a wallet randomly. Your expected gain
    is: ",
     take_a_wallet)
print("Scenario 3: You choose to select a bill before deciding.
   Your expected gain is: ",
     choose_a_bill)
print("Taking into account the maximum expected gain, you should
   ",get_max_scenario(initial_prize,take_a_wallet,choose_a_bill))
```

Bibliography

[1] H. Teimoorinia, R. D. Toyonaga, S. Fabbro, and C. Bottrell, "Comparison of multi-class and binary classification machine learning models in identifying strong gravitational lenses," *Publications of the Astronomical Society of the Pacific*, vol. 132, no. 1010, p. 044501, 2020.

[2] Z. Zhang and C. Sun, "Multi-site structural damage identification using a multi-label classification scheme of machine learning," *Measurement*, vol. 154, p. 107473, 2020.

[3] H. Zhu, G. Liu, M. Zhou, Y. Xie, A. Abusorrah, and Q. Kang, "Optimizing weighted extreme learning machines for imbalanced classification and application to credit card fraud detection," *Neurocomputing*, 2020.

[4] A. Zemmari and J. Benois-Pineau, "Supervised learning problem formulation," in *Deep Learning in Mining of Visual Content*, pp. 5–11, Springer, 2020.

[5] M. Tuda and A. I. Luna-Maldonado, "Image-based insect species and gender classification by trained supervised machine learning algorithms," *Ecological Informatics*, vol. 60, p. 101135, 2020.

[6] M. Tuda and A. I. Luna-Maldonado, "Image-based insect species and gender classification by trained supervised machine learning algorithms," *Ecological Informatics*, vol. 60, p. 101135, 2020.

[7] R. Seifi Majdar and H. Ghassemian, "A probabilistic svm approach for hyperspectral image classification using spectral and texture features," *International Journal of Remote Sensing*, vol. 38, no. 15, pp. 4265–4284, 2017.

[8] P. Rakshit, R. Basu, S. Paul, S. Bhattacharyya, J. Mistri, and I. Nath, "Face detection using support vector mechine with pca," *Available at SSRN 3515989*, 2020.

[9] D. Żurek and M. Pietroń, "Training with reduced precision of a support vector machine model for text classification," *arXiv preprint arXiv:2007.08657*, 2020.

[10] M. Al-Shabi, "A novel cad system for detection and classification of liver cirrhosis using support vector machine and artificial neural network," *IJCSNS*, vol. 19, no. 8, p. 18, 2019.

[11] N. Gruzling, *Linear separability of the vertices of an n-dimensional hypercube*. ProQuest, 2008.

[12] N. G. Polson, S. L. Scott, *et al.*, "Data augmentation for support vector machines," *Bayesian Analysis*, vol. 6, no. 1, pp. 1–23, 2011.

[13] F. Wenzel, T. Galy-Fajou, M. Deutsch, and M. Kloft, "Bayesian nonlinear support vector machines for big data," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 307–322, Springer, 2017.

[14] F. Wenzel, M. Deutsch, T. Galy-Fajou, and M. Kloft, "Scalable approximate inference for the bayesian nonlinear support vector machine,"

[15] S. R. Gunn *et al.*, "Support vector machines for classification and regression," *ISIS technical report*, vol. 14, no. 1, pp. 5–16, 1998.

[16] J. Wakefield, *Spline and Kernel Methods*, pp. 547–595. New York, NY: Springer New York, 2013.

[17] P. H. Swain and H. Hauska, "The decision tree classifier: Design and potential," *IEEE Transactions on Geoscience Electronics*, vol. 15, no. 3, pp. 142–147, 1977.

[18] D. Che, Q. Liu, K. Rasheed, and X. Tao, "Decision tree and ensemble learning algorithms with their applications in bioinformatics," in *Software tools and algorithms for biological systems*, pp. 191–199, Springer, 2011.

[19] M.-C. Wu, S.-Y. Lin, and C.-H. Lin, "An effective application of decision tree to stock trading," *Expert Systems with applications*, vol. 31, no. 2, pp. 270–274, 2006.

[20] L. Rokach and O. Maimon, "Decision trees," in *Data mining and knowledge discovery handbook*, pp. 165–192, Springer, 2005.

[21] L. Rokach and O. Maimon, "Top-down induction of decision trees classifiers-a survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 35, no. 4, pp. 476–487, 2005.

[22] J. R. Quinlan, "Induction of decision trees," *Machine learning*, vol. 1, no. 1, pp. 81–106, 1986.

[23] J. Joyce, "Bayes' theorem," 2003.

[24] B. Efron, "Bayes' theorem in the 21st century," *Science*, vol. 340, no. 6137, pp. 1177–1178, 2013.