
UNIT 10 SUPERVISED LEARNING MODELS

10.0 Introduction

10.1 Expected Learning Outcomes

10.2 Supervised Vs Unsupervised Learning

10.3 Supervised Learning models

10.3.1 K-Nearest Neighbour,

10.3.2 Naïve-Bayes,

10.3.3 Logistic regression,

10.3.4 Decision trees

10.3.5 Random Forest

10.3.6 Support vector Machines

10.4 Summary

10.5 Answers

10.6 References/Further Readings

10.0 INTRODUCTION

Data analysis is essential for turning observations into meaningful predictions and actionable insights. In critical fields such as finance and biomedicine, forecasting future outcomes and accurately classifying data play a vital role in decision-making. Supervised learning formalises this predictive process when the outcome variable is known for a portion of the dataset. It enables the development of models that learn from past observations and apply that knowledge to future scenarios, thereby bridging the gap between raw data and intelligent decision systems.

A strong foundation in mathematics and statistics is essential for understanding supervised learning methodologies. Most machine learning frameworks rely heavily on statistical reasoning, probability theory, and optimization techniques. To effectively grasp supervised learning, one must be familiar with key concepts such as statistical models, loss functions, optimization algorithms, and performance evaluation metrics. These elements collectively determine how well a model learns patterns from data and how accurately it performs in real-world applications.

Supervised learning is fundamentally characterized by the use of labelled training data. For every input instance (X), there exists a corresponding output (Y), and the goal of the algorithm is to learn a mapping function, f , such that $f(X) \approx Y$. Once this relationship is learned, the model can generalize to new, unseen inputs (X_{new}) and predict corresponding outputs (Y_{new}). However, the true effectiveness of a model lies not in how well it performs on training data, but in its ability to generalize to new data. This is measured through generalization error. Poor-quality or biased labels can significantly affect this ability, leading to irreducible errors that cannot be corrected even with advanced algorithms.

Supervised learning problems are broadly categorized into two types:

- Regression, and
- Classification.

Regression deals with predicting continuous values, such as stock prices, temperature, or sales figures. In contrast, *Classification* focuses on assigning discrete labels to data, such as identifying whether an email is spam or not. The choice between regression and classification depends on the nature of the target variable and the problem context.

In practical applications, supervised learning is used across a wide range of domains including image recognition, natural language processing, credit scoring, fraud detection, and medical diagnosis. The typical workflow involves splitting the dataset into training and testing sets, where the training data is used to build the model and the testing data evaluates its predictive performance. Data preprocessing steps such as handling missing values, encoding categorical variables, and normalizing features are critical for ensuring reliable results. Additionally, techniques like cross-validation are employed to fine-tune model parameters and reduce the risk of overfitting, thereby improving model robustness.

Another important aspect of this unit is understanding the distinction between supervised and unsupervised learning. While supervised learning uses labeled data to make predictions, unsupervised learning works with unlabeled data to discover hidden patterns or structures, such as clustering similar data points. This distinction is essential in selecting the appropriate analytical approach based on the problem and data availability.

This unit further introduces several important supervised learning models that are widely used in data analysis. The **K-Nearest Neighbors (KNN)** algorithm is a simple, distance-based method that classifies data based on similarity. The **Naïve Bayes** classifier applies probabilistic reasoning using Bayes' theorem and is particularly effective in text classification tasks. **Logistic Regression** is a widely used statistical technique for classification problems, especially binary outcomes, offering interpretability and efficiency.

The unit also covers **Decision Trees**, which provide a rule-based, hierarchical structure for decision-making and are easy to interpret. Building on this, **Random Forest** is an ensemble technique that combines multiple decision trees to improve prediction accuracy and reduce overfitting. Finally, **Support Vector Machines (SVM)** are introduced as powerful models that determine optimal decision boundaries (hyperplanes) to separate different classes, particularly effective in high-dimensional data spaces.

Overall, this unit provides a comprehensive foundation in supervised learning by integrating theoretical concepts with practical relevance. It equips learners with the knowledge required to understand, implement, and evaluate various supervised learning models, enabling them to transform data into meaningful insights and predictive solutions.

10.1 EXPECTED LEARNING OUTCOMES

After completing this Unit, you will be able:

- To differentiate between supervised and unsupervised learning approaches.
- To comprehend the concept of labeled data and mapping functions in supervised learning.
- To identify and distinguish between regression and classification problems.

- To gain knowledge of key supervised learning models such as KNN, Naïve Bayes, Logistic Regression, Decision Trees, Random Forest, and SVM.

10.2 SUPERVISED VERSUS UNSUPERVISED LEARNING

In the previous section, we learned that Supervised learning mainly focuses on two types of problems based on the type of output (Y): classification and regression. These two approaches help in solving different kinds of real-world prediction problems depending on whether the output is categorical or continuous.

Classification tasks are used when we need to predict a category or label. In this case, the output is discrete, meaning it belongs to a fixed set of classes. For example, an email can be classified as spam or not spam, a user may or may not click on an advertisement, or an image can be identified as a specific object. The main goal of classification models is to correctly assign the right label to each input. These models are designed to improve accuracy and clearly separate different classes using techniques like Logistic Regression or Support Vector Machines.

Regression tasks, on the other hand, are used when the output is a continuous value. This means the result can take any value within a range. Examples include predicting house prices, stock market trends, energy consumption, or disease growth over time. In regression, the aim is to minimize the difference between the predicted and actual values. This is usually done using error measures like Mean Squared Error (MSE) or Mean Absolute Error (MAE).

The supervised learning process follows a step-by-step workflow:

- First, an appropriate algorithm is selected based on the problem.
- Then, the model is trained and evaluated, and improvements are made by adjusting parameters or changing the model structure.

This process continues until the model performs well.

While choosing a model, several factors must be considered, such as training speed, memory usage, accuracy, and interpretability. In fields like healthcare or finance, simpler and more interpretable models are often preferred even if they are slightly less accurate, because understanding the decision process is very important.

Improving a model is an iterative process. It involves managing model complexity to avoid problems like overfitting (model learning too much from training data) or underfitting (model not learning enough). This can be done by tuning parameters, handling imbalanced data, or modifying the model structure, such as limiting the depth of a decision tree. These adjustments help maintain a balance between bias and variance.

Finally, before building any model, data preparation is very important. Steps like handling missing values, normalizing data, and creating useful features help improve model

performance. Good preprocessing ensures that the model works accurately and reliably when applied to real-world data.

Now we will understand the distinction between supervised and unsupervised learning.

In machine learning, data can be analyzed using two main approaches: supervised learning and unsupervised learning.

The primary difference between these approaches lies in the type of data used and the objective of the analysis.

- Supervised learning works with labeled data, where both the input (X) and the correct output (Y) are known. The model learns from this data and tries to predict the correct output for new inputs.
- In contrast, unsupervised learning works with unlabeled data, where only input features are available. Instead of predicting outcomes, the model identifies hidden patterns, structures, or groupings within the data.

To understand this difference more clearly, consider a simple example.

- In supervised learning, if we have emails labeled as “spam” or “not spam,” the model learns from these examples and can classify new emails accordingly.
- On the other hand, in unsupervised learning, if we have a large collection of emails without any labels, the model groups similar emails together based on patterns, such as grouping promotional emails separately from personal or work-related messages.

The objectives of these two approaches are also different.

- Supervised learning focuses on prediction, meaning it tries to determine the correct output for given inputs.
- Unsupervised learning, however, focuses on pattern discovery, aiming to uncover relationships or structures that are not immediately visible.

This difference in objectives makes supervised learning suitable for tasks like classification and regression, while unsupervised learning is more useful for clustering and data exploration.

Another important distinction is related to data requirements.

- Supervised learning requires high-quality labeled data, which can be expensive and time-consuming to collect because it often involves human effort.
- In contrast, unsupervised learning uses raw, unlabeled data, which is easier and cheaper to obtain.

The way performance is evaluated also differs between the two.

- In supervised learning, evaluation is straightforward because predicted outputs can be compared directly with actual outputs using metrics such as accuracy, precision, and recall.

- However, in unsupervised learning, there are no true labels available, so evaluation relies on indirect measures such as how well the data is grouped or how meaningful the discovered patterns are.

In real-life terms,

- supervised learning can be compared to learning with a teacher, where correct answers are provided during training.
- Unsupervised learning, on the other hand, is like self-learning, where patterns are discovered without guidance.

Due to the high cost of labeling data, modern approaches are developed, such as:

- semi-supervised and
- self-supervised learning.

These methods combine small amounts of labeled data with large amounts of unlabeled data to improve learning efficiency and performance. We are not going into their details.

Overall, both supervised and unsupervised learning play important and complementary roles in data analysis.

- Supervised learning is best suited for making accurate predictions when labeled data is available, while
- Unsupervised learning is useful for exploring and understanding hidden patterns in data.

Together, they form the foundation of modern machine learning and intelligent data-driven systems.

So, we learned that in machine learning, one of the main differences lies in how the data is used.

- Supervised learning requires **labeled data**, which means that for every input, the correct output is already known. For example, if we want to build a system to recognize handwritten digits, we need images that are already labeled with the correct number. Using these examples, the model learns patterns like shapes and pixel arrangements and then uses this knowledge to predict the correct output for new data.
- On the other hand, unsupervised learning works with **unlabeled data**, where no correct answers are given. Instead of predicting outcomes, the model tries to find hidden patterns or relationships in the data. For example, it can group news articles into categories like sports or crime, or cluster customers based on similar buying behavior.

It is to be noted that, unlike supervised learning, the unsupervised learning does not have predefined outputs to guide it. The main strength of Unsupervised learning is discovering patterns that may not be obvious. This makes it very useful for tasks like data exploration,

anomaly detection, and reducing data complexity. It is especially helpful when labeling data is difficult, expensive, or time-consuming.

The way we measure success is also different in both approaches.

- In supervised learning, we can directly compare predicted results with actual results using metrics like accuracy, precision, and recall.
- However, in unsupervised learning, since there are no correct answers, we use indirect methods such as how well the data is grouped or how meaningful the patterns are.

So, we learned that supervised learning is best when we want to make accurate predictions using labeled data, while unsupervised learning is useful for discovering hidden patterns in unlabeled data. Both approaches are important and complement each other in solving real-world data problems.

☛ Check Your Progress 1

1. What is the main difference between supervised and unsupervised learning?
.....
.....
2. Which type of learning is used for predicting house prices and why?
.....
.....
3. Give one real-life example of unsupervised learning.
.....
.....
4. Why is data preprocessing important in machine learning?
.....
.....
5. How is supervised learning similar to learning with a teacher?
.....
.....

10.3 SUPERVISED LEARNING MODELS

In the above we learned about Supervised learning and compared it with Unsupervised learning, on the basis of various aspects. Now, in this section we are going to discuss about various Supervised learning models, which are widely used in data analysis, and they learn from labeled data to make predictions or classifications.

These models identify patterns between input features and known outputs, enabling them to classify/predict outcomes for new, unseen data. Different supervised learning algorithms are designed to handle various types of problems such as classification and regression, each with its own strengths and applications. The prominent supervised learning models are as follows:

- *K-Nearest Neighbours (KNN)* is a simple and intuitive algorithm that classifies a data point based on the majority class of its nearest neighbours. It relies on distance

measures such as Euclidean or Manhattan distance and does not require an explicit training phase. Its performance depends on the choice of K and proper feature scaling.

- *Naïve Bayes* is a probabilistic classifier based on Bayes' Theorem. It assumes that features are independent of each other and calculates the probability of each class given the input features. Despite its simplicity, it performs well in applications like spam detection and text classification.
- *Logistic Regression* is a statistical model used mainly for binary classification problems. It estimates the probability of an outcome using a logistic (sigmoid) function and provides interpretable results, making it useful in fields like healthcare and finance.
- *Decision Trees* are tree-structured models that make decisions based on a series of conditions or rules derived from the data. They are easy to understand and interpret but may suffer from overfitting if not properly controlled.
- *Random Forest* is an ensemble learning method that combines multiple decision trees to improve prediction accuracy and reduce overfitting. By aggregating the outputs of several trees, it produces more stable and reliable results.
- *Support Vector Machines (SVM)* are powerful models that classify data by finding the optimal boundary (hyperplane) that separates different classes. They are particularly effective in high-dimensional spaces and can handle both linear and non-linear classification using kernel functions.

These supervised learning models provide a comprehensive toolkit for solving real-world classification problems. The choice of model depends on factors such as the nature of the data, interpretability requirements, and computational efficiency.

But, before going to next section wherein we are going to discuss the technical aspects of the various models of the supervised learning (mentioned above), Let's discuss some technicalities of the workflow of the Supervised learning itself, we are going to start with the Basic Idea of Model Training.

It is to be noted that, in supervised learning, the main goal is to build a model that can correctly predict outputs from inputs. To do this, the model uses something called a **loss function**, which measures how far the predicted value is from the actual value. The training process tries to minimize this error so that predictions become more accurate.

However, if a model learns too much from training data, it may start memorizing noise instead of actual patterns. This problem is called **overfitting**. To avoid this, techniques like **regularization** are used (we learned about regularization in our earlier unit), which control the complexity of the model and help it perform better on new data.

Further, to minimize error, optimization techniques such as **Gradient Descent** are used. This method adjusts model parameters step by step in the direction that reduces error. For large datasets, a faster version called **Stochastic Gradient Descent (SGD)** is used, which works on small batches of data and speeds up training.

Another important concept is the **bias-variance trade-off**. Bias occurs when a model is too simple and cannot capture the real pattern (underfitting), while variance occurs when the model is too complex and reacts too much to small changes in data (overfitting). A good model balances both bias and variance to achieve better performance on new data.

Overall, the goal of supervised learning is to create a model that is accurate, stable, and capable of making reliable predictions on unseen data.

There are various Supervised Learning models to be discussed in this unit, which includes, K-Nearest neighbour, naïve-bayes, Logistic regression, decision trees, Random Forest and Support vector Machines.

In the subsequent section we are going to discuss one of the simplest models i.e. K-Nearest Neighbour (K-NN)

☛ Check Your Progress 2

Q1. What is Supervised Learning and how does it support predictive analytics?

.....
.....

Q2. What are regression and classification problems in supervised learning?

.....
.....

Q3. Differentiate between supervised and unsupervised learning..

.....
.....

Q4. Explain the workflow of supervised learning.

.....
.....

Q5. List and briefly explain any three supervised learning models

.....
.....

Q6. What factors should be considered while selecting a supervised learning model?

.....
.....

10.3.1 K-NEAREST NEIGHBORS (K-NN)

The K-Nearest Neighbors (KNN) algorithm is one of the simplest and most intuitive supervised learning techniques used for both classification and regression tasks. It is a **non-parametric and instance-based learning algorithm**, meaning it does not build an explicit model during training but instead makes predictions based on the stored training data. The core idea behind KNN is that similar data points exist close to each other in the feature space. Therefore, the class or value of a new data point can be determined by examining its nearest neighbors.

In simple terms, KNN works on the principle of “**Similarity**” or “**Closeness**.” For example, in real life, if a new student joins a class and we want to guess their performance, we may compare them with students who have similar study habits or background. Similarly, in a

movie recommendation system, if a user likes certain movies, the system recommends movies liked by users with similar preferences. These are practical examples of how KNN operates.

The working of the KNN algorithm involves the following steps:

- first, choose the value of K (the number of nearest neighbors).
- Then, calculate the distance between the new data point and all points in the training dataset.
- Next, select the K closest data points based on the calculated distance.
- Finally, for classification, assign the class that is most frequent among the neighbors, and for regression, compute the average of the values of the neighbors.

The most commonly used distance measure in KNN is the **Euclidean Distance**, which is given by the formula:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

For higher dimensions, the formula extends as:

$$d = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

Other distance measures such as Manhattan distance or Minkowski distance can also be used depending on the problem.

Note: The **Manhattan distance**, also known as L1 distance, is a method used to measure the distance between two points by calculating the sum of the absolute differences of their coordinates. It is called Manhattan distance because it resembles movement in a city like Manhattan, where one can only travel along streets in horizontal and vertical directions rather than taking a straight diagonal path. The formula for Manhattan distance is given by:

$$d = \sum_{i=1}^n |x_i - y_i|$$

where x_i and y_i represent the values of the i^{th} feature of two data points, and n is the total number of dimensions. The absolute value ensures that all differences are treated as positive distances.

For example, if we consider two points A(2, 3) and B(5, 7), the Manhattan distance between them is calculated as $|2 - 5| + |3 - 7| = 3 + 4 = 7$. This shows that the distance is computed by moving step-by-step along each axis rather than directly.

The **Minkowski Distance** is a more general and flexible distance measure that includes both Manhattan and Euclidean distances as special cases. Its formula is given by:

$$d = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{\frac{1}{p}}$$

where p is a parameter that determines the type of distance, and x_i , y_i , and n is the total number of dimensions.

- When $p = 1$, the Minkowski distance becomes the Manhattan distance, and
- When $p = 2$, it becomes the Euclidean distance.

For example, if we take two points A (2, 3) and B (5, 7) and choose $p = 3$, the distance is calculated as $(|2 - 5|^3 + |3 - 7|^3)^{1/3} = (27 + 64)^{1/3} = 91^{1/3} \approx 4.48$.

This demonstrates how the Minkowski distance can adapt to different scenarios by adjusting the value of p .

Note: It is to be noted that the Lower values of p treat all differences more uniformly, while higher values give more importance to larger differences.

Thus, Minkowski distance provides a flexible framework for measuring similarity between data points in various machine learning applications.

Now, to understand KNN more clearly, consider a simple numerical example. Suppose we have the following training dataset representing students based on two features: study hours and attendance percentage. The target variable indicates whether the student passed (Yes) or failed (No).

Study Hours	Attendance (%)	Result
2	50	No
4	60	No
6	70	Yes
8	80	Yes

Now, suppose a new student has studied for 5 hours and has 65% attendance. We want to predict whether this student will pass or fail using KNN with $K = 3$.

First, we calculate the Euclidean distance between the new student (5, 65) and all training points:

- Distance from (2, 50): $\sqrt{(5 - 2)^2 + (65 - 50)^2} = \sqrt{9 + 225} = \sqrt{234} \approx 15.33$
- Distance from (4, 60): $\sqrt{(5 - 4)^2 + (65 - 60)^2} = \sqrt{1 + 25} = \sqrt{26} \approx 5.10$
- Distance from (6, 70): $\sqrt{(5 - 6)^2 + (65 - 70)^2} = \sqrt{1 + 25} = \sqrt{26} \approx 5.10$
- Distance from (8, 80): $\sqrt{(5 - 8)^2 + (65 - 80)^2} = \sqrt{9 + 225} = \sqrt{234} \approx 15.33$

Next, we select the 3 nearest neighbors (smallest distances):

- (4, 60) → No
- (6, 70) → Yes
- (2, 50) → No (or (8,80) depending on tie, but both far; typically, next nearest is chosen consistently)

Among these 3 neighbors:

- No → 2 times
- Yes → 1 time

Therefore, the predicted class for the new student is **“No” (Fail)**.

The interpretation of this result is that the new student is more similar to students who failed, based on the selected features. Hence, the model predicts that the student is likely to fail. This demonstrates how KNN uses proximity to make decisions rather than learning a mathematical model.

It is very important to understand that the performance of the K-Nearest Neighbors (KNN) algorithm is strongly influenced by the choice of the parameter K, which represents the number of nearest neighbors considered while making a prediction.

- If the value of K is very small, the model may become too sensitive to individual data points or noise, leading to **overfitting**.
- On the other hand, if the value of K is too large, the model may become too generalized and fail to capture important local patterns, leading to **underfitting**.

In addition, KNN is a distance-based algorithm, so the scale of the features plays a major role. If one feature has much larger values than another, it can dominate the distance calculation, which may produce misleading results. Therefore, feature normalization is often necessary.

To understand this clearly, consider the following numerical example. Suppose we want to classify whether a student will **Pass** or **Fail** based on two features: **Study Hours** and **Marks in Internal Assessment**.

The training data is given below:

Student	Study Hours	Internal Marks	Result
A	1	40	Fail
B	2	42	Fail
C	3	45	Fail

Student	Study Hours	Internal Marks	Result
D	4	48	Pass
E	5	50	Pass
F	6	52	Pass
G	7	54	Pass
H	8	90	Fail

Now suppose a new student(X)has joined the class with record of 5 hours of study and 49 as the internal marks i.e. X has following values:

$$X = (5, 49)$$

Now, we want to classify(Using K-NN),whether this student will Pass or Fail using different values of K .

First, let us calculate the Euclidean distance of the new student $X = (5, 49)$ from each training point tabled above.

We know that the Euclidean distance formula is:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Euclidean distance of $X = (5, 49)$ For student A (1, 40):

$$d_A = \sqrt{(5 - 1)^2 + (49 - 40)^2} = \sqrt{16 + 81} = \sqrt{97} \approx 9.85$$

Euclidean distance of $X = (5, 49)$ For student B (2, 42):

$$d_B = \sqrt{(5 - 2)^2 + (49 - 42)^2} = \sqrt{9 + 49} = \sqrt{58} \approx 7.62$$

Euclidean distance of $X = (5, 49)$ For student C (3, 45):

$$d_C = \sqrt{(5 - 3)^2 + (49 - 45)^2} = \sqrt{4 + 16} = \sqrt{20} \approx 4.47$$

Euclidean distance of $X = (5, 49)$ For student D (4, 48):

$$d_D = \sqrt{(5 - 4)^2 + (49 - 48)^2} = \sqrt{1 + 1} = \sqrt{2} \approx 1.41$$

Euclidean distance of $X = (5, 49)$ For student E (5, 50):

$$d_E = \sqrt{(5 - 5)^2 + (49 - 50)^2} = \sqrt{0 + 1} = 1$$

Euclidean distance of $X = (5, 49)$ For student F (6, 52):

$$d_F = \sqrt{(5 - 6)^2 + (49 - 52)^2} = \sqrt{1 + 9} = \sqrt{10} \approx 3.16$$

Euclidean distance of $X = (5, 49)$ For student G (7, 54):

$$d_G = \sqrt{(5 - 7)^2 + (49 - 54)^2} = \sqrt{4 + 25} = \sqrt{29} \approx 5.39$$

Euclidean distance of $X = (5, 49)$ For student H $(8, 90)$:

$$d_H = \sqrt{(5 - 8)^2 + (49 - 90)^2} = \sqrt{9 + 1681} = \sqrt{1690} \approx 41.11$$

Now the distances in ascending order are:

Student	Distance	Result
E	1.00	Pass
D	1.41	Pass
F	3.16	Pass
C	4.47	Fail
G	5.39	Pass
B	7.62	Fail
A	9.85	Fail
H	41.11	Fail

Now let us see what happens for different values of K .

For $K = 1$, only the nearest neighbor is considered. The nearest neighbor is student E, whose class is **Pass**. Therefore, the predicted class is:

Prediction = Pass

This looks correct, but $K = 1$ can be risky. If the nearest point is a noisy or unusual point, the prediction may become unstable. Thus, a very small K can lead to **overfitting**, where the model becomes too dependent on individual training examples.

For $K = 3$, we consider the three nearest neighbors: E, D, and F. Their classes are:

- E $\rightarrow$ Pass
- D $\rightarrow$ Pass
- F $\rightarrow$ Pass

Since all three are Pass, the majority class is:

Prediction = Pass

This result is also Pass, and it is more reliable than $K = 1$ because it is based on a small group of nearby points rather than a single point.

For $K = 5$, we consider E, D, F, C, and G. Their classes are:

- E $\rightarrow$ Pass
- D $\rightarrow$ Pass

- $F \rightarrow \text{Pass}$
- $C \rightarrow \text{Fail}$
- $G \rightarrow \text{Pass}$

Pass appears 4 times and Fail appears 1 time, so:

Prediction = Pass

This is still a good prediction because most nearby students belong to the Pass class.

Now let us consider $K = 7$. The seven nearest neighbors are E, D, F, C, G, B, and A. Their classes are:

- $\text{Pass} \rightarrow E, D, F, G = 4$
- $\text{Fail} \rightarrow C, B, A = 3$

So the predicted class is again:

Prediction = Pass

However, the influence of farther points is increasing. These farther points may not truly represent the local pattern around the new student.

Finally, for $K = 8$, all points are included.

The class counts become:

- $\text{Pass} \rightarrow D, E, F, G = 4$
- $\text{Fail} \rightarrow A, B, C, H = 4$

This creates a tie, and the result becomes ambiguous. In some cases, the algorithm may choose one class based on implementation rules, but clearly the prediction becomes less meaningful. Here, we see that a large K ignores the strong local neighborhood around the new student and starts including distant or irrelevant points such as H, which is very far away. This is an example of **underfitting**, where the model becomes too general and loses important local structure.

The above example shows that choosing the right value of K is very important. A very small K may fit noise and give unstable predictions, while a very large K may smooth out meaningful differences and produce weak predictions. In practice, moderate values such as 3, 5, or 7 are often tested, and the best value is selected using validation techniques.

Choosing the right value of K in KNN is important because it directly affects how well the model performs on new (unseen) data. But the question is—how do we decide which value of K is best?

This is where validation techniques come into play.

Validation techniques help us test different values of K and select the one that gives the best performance, not just on training data but on unseen data as well. The most commonly used method is:

- Train-Test Split or
- Cross-Validation.

First, let us understand the idea using a simple approach called **Train-Test Split**. The dataset is divided into two parts:

- **Training Data (e.g., 70–80%)**: used to build the model
- **Testing Data (e.g., 20–30%)**: used to evaluate performance

Now, we try different values of K (say 1, 3, 5, 7, 9) and check how well each performs on the test data.

Example 1: Suppose we try different values of K and get the following accuracy:

K Value	Accuracy (%)
1	85
3	90
5	92
7	88
9	83

Solution: From this table, we can see that:

- $K = 1$ gives good accuracy but may overfit
- $K = 5$ gives the **highest accuracy (92%)**
- Larger K values start reducing performance

So, we select:

$K = 5$ (best performance)

This method works, but it depends on one split of data.

To make it more reliable, we use a better technique called **Cross-Validation**.

Cross-Validation (More Reliable Method)

In **K-Fold Cross-Validation**, the dataset is divided into K equal parts (called folds). For example, if $K = 5$:

- Split data into 5 parts
- Train the model on 4 parts and test on 1 part
- Repeat this process 5 times (each time using a different test part)
- Take the **average accuracy**

Now, we repeat this entire process for different values of K (neighbors). The results are:

Result :

K (neighbors)	Avg. Accuracy (%)
1	84
3	89

K (neighbors)	Avg. Accuracy (%)
5	91
7	87
9	82

Here again, the highest accuracy is for:

$$K = 5$$

So, cross-validation confirms that $K = 5$ is the best choice.

The interpretation of different values of K in the K-Nearest Neighbors (KNN) algorithm is very important for understanding model behavior.

- When the value of K is very small, such as $K = 1$, the model relies heavily on the nearest data point. This makes the model highly sensitive to noise or outliers in the dataset, which can lead to **overfitting**, where the model performs well on training data but poorly on new data.
- On the other hand, when the value of K is very large, such as $K = 9$, the model considers too many neighboring points. As a result, it may ignore important local patterns in the data and produce overly generalized predictions, leading to **underfitting**.
- A moderate value of K , such as 3, 5, or 7, usually provides a good balance because it considers enough neighbors to make stable predictions while still capturing meaningful patterns. This results in better **generalization** on unseen data.

The key idea behind using validation techniques is to determine which value of K performs best on data that the model has not seen before. Instead of selecting K randomly, a systematic approach is followed:

- Multiple values of K are tested
- The performance of each value is measured using validation methods
- The value that gives the best average accuracy is selected

So, we learned that the validation techniques such as train-test split and cross-validation help in choosing an appropriate value of K . These techniques ensure that the selected K :

- Avoids overfitting (caused by very small K)
- Avoids underfitting (caused by very large K)
- Provides the best predictive performance on unseen data

Thus, validation plays a crucial role in finding the optimal balance, making the KNN model more reliable, stable, and accurate.

Now let us understand why **feature scaling or normalization** is also important in KNN. Suppose instead of the previous dataset, we have two features with very different scales: **Study Hours** ranging from 1 to 8, and **Annual Family Income** ranging from 40,000 to 90,000. If we directly calculate distances, the income feature will dominate because its numerical values are much larger.

For example, consider two students, $P = (5, 40000)$ and $Q = (6, 90000)$

The Euclidean distance is:

$$d = \sqrt{(5 - 6)^2 + (40000 - 90000)^2}$$
$$d = \sqrt{1 + 2500000000} \approx 50000$$

Here, the difference in study hours is only 1, but it becomes negligible compared to the huge income difference. As a result, the algorithm pays almost no attention to study hours, even if that feature is very important for prediction. This is why normalization is needed.

A common normalization method is **Min-Max Normalization**, given by:

$$x' = \frac{x - x_{\min}}{x_{\max} - x_{\min}}$$

This transforms all feature values into a common scale, usually between 0 and 1. After normalization, both study hours and income contribute more fairly to the distance calculation. This improves the reliability of KNN predictions.

Now let us normalize both features.

For **Study Hours**, the minimum and maximum values are:

$$x_{\min} = 1, \text{ and } x_{\max} = 8$$

So the range is: $8 - 1 = 7$

For **Annual Family Income**, the minimum and maximum values are:

$$x_{\min} = 40000, \quad x_{\max} = 90000$$

So the range is: $90000 - 40000 = 50000$

Now normalize student **P** = (5, 40000).

For study hours:

$$5' = \frac{5 - 1}{8 - 1} = \frac{4}{7} \approx 0.571$$

For income:

$$40000' = \frac{40000 - 40000}{90000 - 40000} = \frac{0}{50000} = 0$$

So the normalized coordinates of **P** are:

$$P' = (0.571, 0)$$

Now normalize student **Q** = (6, 90000).

For study hours:

$$6' = \frac{6 - 1}{8 - 1} = \frac{5}{7} \approx 0.714$$

For income:

$$90000' = \frac{90000 - 40000}{90000 - 40000} = \frac{50000}{50000} = 1$$

So the normalized coordinates of **Q** are:

$$Q' = (0.714, 1)$$

Now calculate Euclidean distance using the normalized values:

$$\begin{aligned} d(P', Q') &= \sqrt{(0.571 - 0.714)^2 + (0 - 1)^2} \\ &= \sqrt{(-0.143)^2 + (-1)^2} \\ &= \sqrt{0.0204 + 1} \\ &= \sqrt{1.0204} \approx 1.01 \end{aligned}$$

Now the distance is around **1.01**, and both features contribute in a balanced way.

The study-hours difference is no longer ignored.

To see why this matters in KNN, let us take a simple classification example.

Assume we want to predict whether a student will **Pass** or **Fail** based on Study Hours and Family Income. Consider the following training data:

Student	Study Hours	Income	Result
A	5	40000	Pass
B	6	90000	Fail
C	4	42000	Pass

Now suppose a new student is:

$$X = (5, 45000)$$

We want to classify this student using KNN.

Classify without normalization:

Distance from **X** to **A**:

$$\begin{aligned} d(X, A) &= \sqrt{(5 - 5)^2 + (45000 - 40000)^2} \\ &= \sqrt{0 + 5000^2} = 5000 \end{aligned}$$

Distance from **X** to **B**:

$$\begin{aligned} d(X, B) &= \sqrt{(5 - 6)^2 + (45000 - 90000)^2} \\ &= \sqrt{1 + 45000^2} \\ &= \sqrt{1 + 2025000000} \approx 45000 \end{aligned}$$

Distance from **X** to **C**:

$$d(X, C) = \sqrt{(5 - 4)^2 + (45000 - 42000)^2}$$

$$= \sqrt{1 + 3000^2}$$

$$= \sqrt{1 + 9000000} \approx 3000$$

So nearest neighbours are dominated by income differences. Study hours have almost no effect.

Now, normalize the data and Classify:

For the new student $X = (5, 45000)$:

Study hours:

$$5' = \frac{5 - 1}{7} = \frac{4}{7} \approx 0.571$$

Income:

$$45000' = \frac{45000 - 40000}{50000} = \frac{5000}{50000} = 0.1$$

So,

$$X' = (0.571, 0.1)$$

For student A = (5, 40000):

$$A' = (0.571, 0)$$

For student B = (6, 90000):

$$B' = (0.714, 1)$$

For student C = (4, 42000):

Study hours:

$$4' = \frac{4 - 1}{7} = \frac{3}{7} \approx 0.429$$

Income:

$$42000' = \frac{42000 - 40000}{50000} = \frac{2000}{50000} = 0.04$$

So,

$$C' = (0.429, 0.04)$$

Now compute normalized distances.

Distance from X' to A' :

$$d(X', A') = \sqrt{(0.571 - 0.571)^2 + (0.1 - 0)^2}$$

$$= \sqrt{0 + 0.01} = 0.1$$

Distance from X' to B' :

$$\begin{aligned}d(X', B') &= \sqrt{(0.571 - 0.714)^2 + (0.1 - 1)^2} \\&= \sqrt{(-0.143)^2 + (-0.9)^2} \\&= \sqrt{0.0204 + 0.81} \\&= \sqrt{0.8304} \approx 0.911\end{aligned}$$

Distance from X' to C' :

$$\begin{aligned}d(X', C') &= \sqrt{(0.571 - 0.429)^2 + (0.1 - 0.04)^2} \\&= \sqrt{(0.142)^2 + (0.06)^2} \\&= \sqrt{0.0202 + 0.0036} \\&= \sqrt{0.0238} \approx 0.154\end{aligned}$$

So after normalization:

- $d(X', A') = 0.1$
- $d(X', C') = 0.154$
- $d(X', B') = 0.911$

Now the nearest students are **A** and **C**, both labeled **Pass**. Hence, KNN will classify the new student as:

Predicted Class = Pass

This example clearly shows why normalization is important in KNN. Without normalization, the large income values dominate the distance formula, and smaller-scale features like study hours are almost ignored. This can give misleading neighbors and wrong predictions. After normalization, both study hours and income are brought to the same scale, so each feature contributes fairly to the distance calculation.

Thus, normalization improves the reliability and fairness of KNN predictions.

In general:

- KNN depends fully on distance
- Distance gets distorted when features have different scales
- Normalization puts all features on a common scale
- This helps KNN identify truly similar neighbors

So, before applying KNN, feature scaling is usually an essential preprocessing step.

The interpretation of these results is straightforward. The value of K controls how local or global the decision is. Small K values focus on very close neighbors and may capture noise, causing overfitting. Large K values include too many neighbors, even distant ones, and may ignore real patterns, causing underfitting. At the same time, because KNN depends entirely

on distance, features must be on comparable scales; otherwise, one feature can unfairly dominate the prediction. Thus, good performance in KNN depends on two important choices: selecting an appropriate value of K and properly normalizing the data.

So, we learned that the KNN algorithm is simple and powerful, but its success depends heavily on careful parameter selection and data preparation. The above numerical example clearly shows that the choice of K affects the balance between overfitting and underfitting, while normalization ensures that all features contribute fairly in the distance calculation. These considerations are essential for building an accurate and meaningful KNN model.

Also, the performance of KNN depends heavily on the choice of K . A small value of K may lead to overfitting, where the model becomes too sensitive to noise. A large value of K may lead to underfitting, where important patterns are ignored. Additionally, KNN is sensitive to the scale of data, so feature normalization is often required.

There are some advantages and limitations to KNN, they are as follows:

KNN has several advantages.

- It is simple to understand and implement,
- requires no training phase, and
- can handle both classification and regression tasks.

However, KNN also has limitations.

- It is computationally expensive for large datasets because it stores all training data and calculates distances for every prediction.
- It is also sensitive to irrelevant features and noisy data.

So, in this section, we learned that the K-Nearest Neighbors algorithm is a powerful yet simple supervised learning technique that relies on similarity and distance. It is widely used in applications such as recommendation systems, pattern recognition, and classification problems. Despite its simplicity, careful selection of parameters and preprocessing steps are essential to achieve accurate and reliable results.

☛ Check Your Progress 3

Q1. What is the K-Nearest Neighbors (KNN) algorithm? Explain its basic principle.

.....
.....

Q2. Explain the working steps of the KNN algorithm.

.....
.....

Q3. What is the role of the parameter K in KNN?

.....
.....

Q4. Why is feature scaling important in KNN?

.....
.....

Q5. What are the advantages and limitations of KNN?

10.3.2 NAÏVE BAYES ALGORITHM

The Naïve Bayes algorithm is a popular and simple supervised learning technique used mainly for classification problems. It is based on probability theory, specifically on **Bayes' Theorem**, and is widely used in applications such as spam detection, sentiment analysis, medical diagnosis, and document classification. The term “naïve” comes from the assumption that all features are **independent of each other**, which is rarely true in real life but works surprisingly well in practice.

In simple terms, Naïve Bayes answers the question: “*Given some features, what is the probability that a data point belongs to a particular class?*” For example, in email filtering, if an email contains words like “offer,” “free,” and “discount,” the algorithm calculates the probability that the email is spam based on these features and assigns the class with the highest probability.

The core of the Naïve Bayes algorithm is based on Bayes' Theorem, which is expressed as:

$$P(C | X) = \frac{P(X | C) \cdot P(C)}{P(X)}$$

where $P(C | X)$ is the **posterior probability** (probability of class C given input X), $P(X | C)$ is the **likelihood** (probability of features given the class), $P(C)$ is the **prior probability** of the class, and $P(X)$ is the probability of the input features. Since $P(X)$ is constant for all classes, the formula is often simplified as:

$$P(C | X) \propto P(X | C) \cdot P(C)$$

Under the naïve assumption of independence, the likelihood can be written as:

$$P(X | C) = P(x_1 | C) \cdot P(x_2 | C) \cdot \dots \cdot P(x_n | C)$$

This simplification makes the computation efficient even for large datasets.

To understand this clearly, consider a real-life example of email classification. Suppose we want to classify emails as **Spam** or **Not Spam** based on the presence of two words: “Free” and “Offer.” The training data is as follows:

Email	Free	Offer	Class
1	Yes	Yes	Spam
2	Yes	No	Spam
3	No	Yes	Not Spam
4	No	No	Not Spam

First, we calculate prior probabilities:

$$P(\text{Spam}) = \frac{2}{4} = 0.5, \quad P(\text{Not Spam}) = \frac{2}{4} = 0.5,$$

Now, calculate likelihoods:

For Spam:

$$P(\text{Free} = \text{Yes} \mid \text{Spam}) = \frac{2}{2} = 1,$$

$$P(\text{Offer} = \text{Yes} \mid \text{Spam}) = \frac{1}{2} = 0.5$$

For Not Spam:

$$P(\text{Free} = \text{Yes} \mid \text{Not Spam}) = \frac{0}{2} = 0,$$

$$P(\text{Offer} = \text{Yes} \mid \text{Not Spam}) = \frac{1}{2} = 0.5$$

Now suppose we receive a new email with features:

$$X = (\text{Free} = \text{Yes}, \text{Offer} = \text{Yes})$$

We calculate probabilities:

For Spam:

$$P(\text{Spam} \mid X) \propto 1 \times 0.5 \times 0.5 = 0.25$$

For Not Spam:

$$P(\text{Not Spam} \mid X) \propto 0 \times 0.5 \times 0.5 = 0$$

Since $P(\text{Spam} \mid X) > P(\text{Not Spam} \mid X)$, the email is classified as:Spam

The interpretation of this result is that the presence of the word “Free” strongly indicates spam, and hence the algorithm assigns the email to the spam category. This example shows how Naïve Bayes uses probabilities to make decisions.

Let's understand the Naïve Bayes Algorithm with one more Numerical example

Example 2: A medical diagnostic system is designed to predict whether a patient is suffering from **Flu** or **No Flu** based on three symptoms: **Fever**, **Cough**, and **Body Ache**. From previously collected patient records, the following training data are available:

Patient	Fever	Cough	Body Ache	Class
1	Yes	Yes	Yes	Flu

Patient	Fever	Cough	Body Ache	Class
2	Yes	Yes	No	Flu
3	Yes	No	Yes	Flu
4	No	Yes	Yes	Flu
5	Yes	Yes	Yes	Flu
6	No	Yes	No	No Flu
7	No	No	Yes	No Flu
8	No	No	No	No Flu
9	Yes	No	No	No Flu
10	No	Yes	No	No Flu

Using the **Naïve Bayes Algorithm**, classify a new patient with the following symptoms:

- **Fever = Yes**
- **Cough = Yes**
- **Body Ache = No**

Also show all steps, equations, and interpret the result.

Solution:

Step 1: Understand the Objective

We need to predict the class of a new patient:

$$X = (\text{Fever} = \text{Yes}, \text{Cough} = \text{Yes}, \text{Body Ache} = \text{No})$$

The two possible classes are:

- $C_1 = \text{Flu}$
- $C_2 = \text{No Flu}$

Naïve Bayes tells us to calculate:

$$P(C_k | X) = \frac{P(X | C_k) P(C_k)}{P(X)}$$

Since $P(X)$ is same for both classes, for comparison we use:

$$P(C_k | X) \propto P(X | C_k) P(C_k)$$

Step 2: Equation Used in Naïve Bayes

For multiple features, Naïve Bayes assumes that all features are conditionally independent given the class. Therefore,

$$P(X | C_k) = P(x_1 | C_k) \cdot P(x_2 | C_k) \cdot P(x_3 | C_k)$$

So the decision rule becomes:

$$P(C_k | X) \propto P(C_k) \cdot P(x_1 | C_k) \cdot P(x_2 | C_k) \cdot P(x_3 | C_k)$$

Step 3: Meaning of Variables

- $P(C_k)$: Prior probability of class C_k

- $P(x_i | C_k)$: Conditional probability of feature x_i given class C_k
- $P(C_k | X)$: Posterior probability of class C_k after observing the symptoms
- X : New observation to be classified
- x_1, x_2, x_3 : Individual features of X

Here,

- x_1 = Fever = Yes
- x_2 = Cough = Yes
- x_3 = Body Ache = No

Step 4: Find Prior Probabilities

From the table:

- Total patients = 10
- Number of Flu cases = 5
- Number of No Flu cases = 5

Therefore,

$$P(\text{Flu}) = \frac{5}{10} = 0.5$$

$$P(\text{No Flu}) = \frac{5}{10} = 0.5$$

Step 5: Find Conditional Probabilities for Class = Flu

Consider only the rows where Class = Flu:

Patient	Fever	Cough	Body Ache	Class
1	Yes	Yes	Yes	Flu
2	Yes	Yes	No	Flu
3	Yes	No	Yes	Flu
4	No	Yes	Yes	Flu
5	Yes	Yes	Yes	Flu

Total Flu cases = 5

Now calculate the required probabilities:

1. Probability of Fever = Yes given Flu

Number of Flu patients having Fever = Yes = 4

$$P(\text{Fever=Yes} | \text{Flu}) = \frac{4}{5} = 0.8$$

2. Probability of Cough = Yes given Flu

Number of Flu patients having Cough = Yes = 4

$$P(\text{Cough=Yes} | \text{Flu}) = \frac{4}{5} = 0.8$$

3. Probability of Body Ache = No given Flu

Number of Flu patients having Body Ache = No = 1

$$P(\text{Body Ache=No} \mid \text{Flu}) = \frac{1}{5} = 0.2$$

Step 6: Find Conditional Probabilities for Class = No Flu

Consider only the rows where Class = No Flu:

Patient	Fever	Cough	Body Ache	Class
6	No	Yes	No	No Flu
7	No	No	Yes	No Flu
8	No	No	No	No Flu
9	Yes	No	No	No Flu
10	No	Yes	No	No Flu

Total No Flu cases = 5

Now calculate the required probabilities:

1. Probability of Fever = Yes given No Flu

Number of No Flu patients having Fever = Yes = 1

$$P(\text{Fever=Yes} \mid \text{No Flu}) = \frac{1}{5} = 0.2$$

2. Probability of Cough = Yes given No Flu

Number of No Flu patients having Cough = Yes = 2

$$P(\text{Cough=Yes} \mid \text{No Flu}) = \frac{2}{5} = 0.4$$

3. Probability of Body Ache = No given No Flu

Number of No Flu patients having Body Ache = No = 4

$$P(\text{Body Ache=No} \mid \text{No Flu}) = \frac{4}{5} = 0.8$$

Step 7: Compute Posterior Score for Each Class

We now calculate the Naïve Bayes score for each class.

For Flu

$$P(\text{Flu} \mid X) \propto P(\text{Flu}) \cdot P(\text{Fever=Yes} \mid \text{Flu}) \cdot P(\text{Cough=Yes} \mid \text{Flu}) \cdot P(\text{Body Ache=No} \mid \text{Flu})$$

$$P(\text{Flu} \mid X) \propto 0.5 \times 0.8 \times 0.8 \times 0.2$$

$$P(\text{Flu} \mid X) \propto 0.064$$

For No Flu

$$P(\text{No Flu} | X) \propto P(\text{No Flu}) \cdot P(\text{Fever}=\text{Yes} | \text{No Flu}) \cdot P(\text{Cough}=\text{Yes} | \text{No Flu}) \cdot P(\text{Body Ache}=\text{No} | \text{No Flu})$$

$$P(\text{No Flu} | X) \propto 0.5 \times 0.2 \times 0.4 \times 0.8$$

$$P(\text{No Flu} | X) \propto 0.032$$

Step 8: Compare the Results

- Score for Flu = **0.064**
- Score for No Flu = **0.032**

Since,

$$0.064 > 0.032$$

therefore, the new patient is classified as: **Flu**

That means, a patient with symptoms **Fever = Yes, Cough = Yes, Body Ache = No** is classified as: **Flu**

The Interpretation of the result means that based on the available training data, the combination of symptoms shown by the new patient is **more likely to occur in patients having Flu than in patients having No Flu**. Even though the symptom **Body Ache = No** is less common among Flu patients, the other two symptoms, **Fever = Yes** and **Cough = Yes**, strongly support the Flu class. Hence, after multiplying all probabilities, the posterior score of Flu becomes higher than that of No Flu.

In practical terms, Naïve Bayes does not say with absolute certainty that the patient has Flu. Rather, it says that **among the available classes, Flu is the more probable class** for this symptom pattern.

Short Summarized Calculation of above Numerical:

In exam, you can write the answer in this compact form:

$$P(C_k | X) \propto P(C_k) \prod P(x_i | C_k)$$

For Flu:

$$0.5 \times \frac{4}{5} \times \frac{4}{5} \times \frac{1}{5} = 0.064$$

For No Flu:

$$0.5 \times \frac{1}{5} \times \frac{2}{5} \times \frac{4}{5} = 0.032$$

Since $0.064 > 0.032$, the patient belongs to class **Flu**.

10.3.2.1 TYPES OF NAÏVE BAYES ALGORITHM

Now we extend our discussion to explore further details on Naïve Bayes algorithm, In practical scenarios, Naïve Bayes is often implemented in different forms such as:

- Gaussian Naïve Bayes (for continuous data),
- Multinomial Naïve Bayes (for text data), and
- Bernoulli Naïve Bayes (for binary features).

One common issue is the occurrence of zero probability, which can be handled using techniques like **Laplace smoothing**, where a small value is added to all probabilities to avoid multiplication by zero.

A brief introduction to each of the Naïve Bayes algorithm, which are implemented as different variants, depending on the nature of data, such as Gaussian, Multinomial, and Bernoulli Naïve Bayes is as follows:

1. Gaussian Naïve Bayes (for Continuous Data) : Gaussian Naïve Bayes is used when the input features are continuous in nature, such as height, weight, marks, or income. It assumes that the values of these features follow a normal (Gaussian) distribution. Instead of counting frequencies, this method uses the mean and variance of the data to estimate probabilities. The probability of a feature value given a class is computed using the Gaussian distribution formula:

$$P(x | C_k) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left(-\frac{(x-\mu)^2}{2\sigma^2} \right)$$

Where:

- μ = mean of the feature
- σ^2 = variance
- x = observed value

For example, consider a classification problem where we want to determine whether a person is Fit or Unfit based on their weight. Suppose the average weight of Fit individuals is 65 kg with a variance of 25, and for Unfit individuals, the average weight is 85 kg with a variance of 36. If a new person has a weight of 70 kg, we substitute this value into the Gaussian formula for both classes. The class for which the probability value is higher is selected as the predicted class. This approach essentially checks which distribution curve the given value fits better.

2. Multinomial Naïve Bayes (for Text Data): Multinomial Naïve Bayes is mainly used for text-based applications where features represent the frequency or count of words. It is widely applied in spam detection, sentiment analysis, and document classification. In this method, the probability is calculated based on how often a word appears in a particular class compared to the total number of words in that class.

For instance, consider the problem of classifying emails as Spam or Not Spam. Suppose in Spam emails, the word “Offer” appears 50 times out of a total of 100 words, giving a probability of 0.5. In contrast, in Not Spam emails, the same word appears only 5 times out of 100 words, resulting in a probability of 0.05. Now, if a new email contains the word “Offer” multiple times, the probability of it being Spam increases significantly. Thus, words that frequently occur in a class strongly influence the classification decision.

3. Bernoulli Naïve Bayes (for Binary Features): Bernoulli Naïve Bayes is used when the features are binary, meaning they can take only two values: 0 or 1, representing absence or presence of a feature. Unlike Multinomial Naïve Bayes, it does not consider how many times a feature appears but only whether it appears or not.

For example, consider a sentiment analysis problem where we classify a review as Positive or Negative based on the presence of certain words. If the word “Good” appears in a review, it is represented as 1; otherwise, it is 0. Suppose the probability of the word “Good” appearing in Positive reviews is 0.8, while in Negative reviews it is only 0.2. If a new review contains the word “Good,” the model will assign a higher probability to the Positive class. Thus, the presence or absence of specific features plays a crucial role in classification.

Each variant handles features differently, making the algorithm flexible and widely applicable. Another important practical issue is the **zero-probability problem**, which is addressed using **Laplace smoothing**, a brief of Laplace smoothing is as follows:

Laplace Smoothing (Handling Zero Probability): In practical applications of Naïve Bayes, a common issue arises when a feature value does not appear in the training data for a particular class. In such cases, the conditional probability becomes zero. Since Naïve Bayes multiplies probabilities, even a single zero value makes the entire probability zero, leading to incorrect predictions.

To overcome this problem, Laplace smoothing is applied. It modifies the probability calculation by adding a small constant (usually 1) to each count:

$$P(x_i | C_k) = \frac{\text{Count} + 1}{\text{Total} + n}$$

where Count is the frequency of the feature, Total is the total number of observations in the class, and n is the number of possible feature values.

For example, suppose in Spam emails the word “Lottery” has never appeared. Without smoothing, its probability would be zero, which would incorrectly eliminate the Spam class if this word appears in a new email. Using Laplace smoothing, the probability becomes:

$$P = \frac{0 + 1}{100 + 2} = \frac{1}{102}$$

This ensures that no probability becomes zero and the model can handle unseen data more effectively.

Now, we are in position to conclude this section, and prior to this we need to address the advantages and limitations of Naïve Bayes algorithm, they are as follows:

The Naïve Bayes algorithm offers several advantages.

- It is simple to understand and easy to implement.
- It is computationally efficient and works well even with large datasets.
- It performs particularly well in high-dimensional problems such as text classification.
- It also requires relatively less training data compared to many other algorithms.

However, it also has some limitations.

- The assumption of feature independence is often unrealistic, which can affect accuracy in some cases.
- It may not perform well when features are highly correlated.
- Additionally, if the dataset is very small or biased, the probability estimates may not be reliable.

This section concludes with the understanding that the Naïve Bayes algorithm is a powerful yet simple probabilistic classifier that works effectively in many real-world applications. Despite its simplifying assumptions, it provides fast and reasonably accurate predictions, making it a valuable tool in supervised learning. Apart from this, it is important to understand that the choice of Naïve Bayes variant depends on the type of data being used. Gaussian Naïve Bayes is suitable for continuous numerical data, Multinomial Naïve Bayes is ideal for frequency-based data such as text, and Bernoulli Naïve Bayes is appropriate for binary features. Laplace smoothing acts as a practical enhancement that improves model robustness by preventing zero probability issues. Together, these variations make Naïve Bayes a simple yet powerful algorithm applicable across a wide range of real-world problems.

☛ Check Your Progress 4:

Q1. What is the Naïve Bayes algorithm? Why is it called “naïve”?

.....

Q2. State Bayes’ Theorem and explain its components.

.....

Q3. Explain the working of the Naïve Bayes classifier.

.....

Q4. What are the advantages of the Naïve Bayes algorithm?

.....

Q5. What are the limitations of the Naïve Bayes algorithm?

10.3.3 LOGISTIC REGRESSION

Logistic Regression is one of the most fundamental and widely used supervised learning algorithms for solving **classification problems**, especially when the output is binary in nature (such as Yes/No, 0/1, True/False). Despite its name, it is not a regression algorithm in the traditional sense; instead, it is used to estimate the probability that a given input belongs to a particular class.

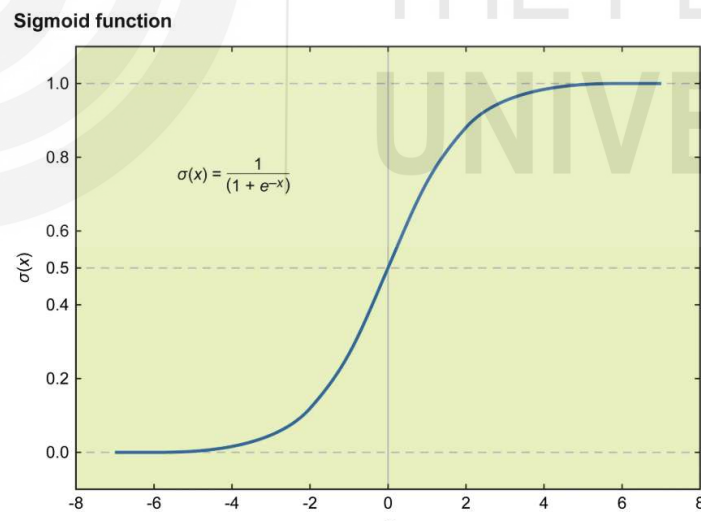
At its core, Logistic Regression builds a relationship between the input features and the probability of an outcome using a mathematical function called the **Sigmoid (Logistic) Function**.

Important Note: The **Sigmoid Function** is a mathematical function used in Logistic Regression to convert any real-valued input into a probability value between **0 and 1**.

Mathematical Expression for Sigmoid Function is: $\sigma(z) = \frac{1}{1+e^{-z}}$

Where:

- z = linear combination of inputs (e.g., $\beta_0 + \beta_1 x$)
- e = Euler's number (≈ 2.718)
- $\sigma(z)$ = output probability between 0 and 1



The sigmoid curve shown above is an S-shaped smooth curve that plays a crucial role in transforming values in Logistic Regression. It maps any real-valued input z into a range between 0 and 1, making it suitable for probability interpretation. When the value of z is very large and negative, the output of the sigmoid function approaches 0, indicating a very low probability. When $z = 0$, the sigmoid function gives an output of 0.5, which represents an uncertain or neutral state. As the value of z becomes large

and positive, the output of the sigmoid function approaches 1, indicating a very high probability.

Mathematically, this behaviour can be observed as $\sigma(0) = 0.5$, $\sigma(z) \rightarrow 1$ as $z \rightarrow +\infty$, and $\sigma(z) \rightarrow 0$ as $z \rightarrow -\infty$.

The sigmoid function can be understood as a probability converter that takes any input value, such as marks, income, or a score, and converts it into a probability between 0 and 1. For example, when $z = -2$, the sigmoid value is approximately 0.12, which indicates a low probability. When $z = 0$, the output is exactly 0.5, representing uncertainty. When $z = 2$, the sigmoid value becomes approximately 0.88, indicating a high probability. This smooth transition from low to high values allows the model to make gradual and interpretable predictions.

In the context of Logistic Regression, the sigmoid function plays a central role by converting the linear combination of input features into a probability. This probability is then used to classify observations based on a threshold value, typically 0.5. Additionally, the sigmoid function is smooth and differentiable, which makes it suitable for optimization techniques used during model training. Overall, the sigmoid function acts as a bridge between linear models and probability theory, ensuring that predictions remain meaningful, interpretable, and bounded within a valid range, thereby making Logistic Regression effective for classification problems.

Unlike linear regression, which can produce any real-valued output, Logistic Regression ensures that the predicted values lie between 0 and 1, making them interpretable as probabilities.

The mathematical expression of Logistic Regression is given by:

$$P(Y = 1 | X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n)}}$$

Where:

- $P(Y = 1 | X)$: Probability that the output belongs to class 1
- e : Euler's number (≈ 2.718)
- β_0 : Intercept (bias term)
- $\beta_1, \beta_2, \dots, \beta_n$: Model coefficients
- $x_1, x_2, \dots, x_n$: Input features

The expression inside the exponent is a linear combination of inputs, but the sigmoid function transforms it into a probability between 0 and 1.

The sigmoid function produces an S-shaped curve. When the linear combination of inputs is very large, the output approaches 1; when it is very small, the output approaches 0. A threshold (commonly 0.5) is used to convert probabilities into class labels.

For example,

- If $P \geq 0.5$, classify as 1

- If $P < 0.5$, classify as 0

This makes Logistic Regression very useful for decision-making problems.

If we talk about a real-life example, consider a bank that wants to predict whether a customer will **default on a loan** based on their income.

- Input: Income
- Output: Default (1) or No Default (0)

Suppose the model learns the relationship such that lower income increases the probability of default. Logistic Regression will output a probability (e.g., 0.75), which indicates a high chance of default.

In terms of the numerical example, let us consider a simple Logistic Regression model:

- Intercept (β_0) = -4
- Coefficient (β_1) = 0.05
- Feature: Income (in thousands)

We want to predict the probability of default for a person earning ₹60,000.

Solution:

Step 1: Compute Linear Combination

$$z = \beta_0 + \beta_1 x$$

$$z = -4 + (0.05 \times 60) = -4 + 3 = -1$$

Step 2: Apply Sigmoid Function

$$P = \frac{1}{1 + e^{-z}} = \frac{1}{1 + e^1}$$

$$P = \frac{1}{1 + 2.718} \approx 0.268$$

Step 3: Classification

Since $P = 0.268 < 0.5$, the prediction is: No Default

The model predicts that the probability of default is approximately **26.8%**, which is relatively low. Therefore, the customer is classified as **non-risky**. This shows how Logistic Regression not only provides a class label but also gives a meaningful probability that can assist in decision-making.

Let's understand the Logistic Regression algorithm in more detail through a Numerical

Example 4: A bank wants to predict whether a customer will **purchase a term deposit** or **not purchase it** based on the customer's **monthly income**. A Logistic Regression model has already been trained, and the obtained model is:

$$z = \beta_0 + \beta_1 x$$

where,

$$z = -6 + 0.08x$$

Here, x represents monthly income in thousand rupees.

Using this Logistic Regression model, predict whether a customer with monthly income of ₹90,000 will purchase the term deposit or not. Use the classification threshold as 0.5. Also compute the probability and interpret the result.

Solution:

We know that formula for Logistic Regression gives the probability of belonging to class 1 as:

$$P(Y = 1 | X) = \frac{1}{1 + e^{-z}}$$

where,

$$z = \beta_0 + \beta_1 x$$

Meaning of Variables

- Y : output class
- $Y = 1$: customer will purchase the term deposit
- $Y = 0$: customer will not purchase the term deposit
- X : input feature
- x : value of the input feature (monthly income)
- β_0 : intercept or bias term
- β_1 : coefficient of income
- z : linear combination of input and coefficients
- $P(Y = 1 | X)$: probability that the customer will purchase the term deposit

Given Data

- Intercept: $\beta_0 = -6$
- Coefficient of income: $\beta_1 = 0.08$
- Income of customer: $x = 90$ thousand rupees

So, the model is:

$$z = -6 + 0.08x$$

Step 1: Calculate the Linear Value z

Substitute $x = 90$ into the equation:

$$z = -6 + 0.08(90)$$

$$z = -6 + 7.2$$

$$z = 1.2$$

Step 2: Apply the Sigmoid Function

Now use the Logistic Regression equation:

$$P(Y = 1 | X) = \frac{1}{1 + e^{-z}}$$

Substitute $z = 1.2$:

$$P(Y = 1 | X) = \frac{1}{1 + e^{-1.2}}$$

We know that:

$$e^{-1.2} \approx 0.3010$$

Therefore,

$$P(Y = 1 | X) = \frac{1}{1 + 0.3010}$$

$$P(Y = 1 | X) = \frac{1}{1.3010}$$

$$P(Y = 1 | X) \approx 0.7686$$

So,

$$P(Y = 1 | X) \approx 0.769$$

Step 3: Apply Decision Rule

Decision rule in Logistic Regression:

- If $P(Y = 1 | X) \geq 0.5$, classify as **1**
- If $P(Y = 1 | X) < 0.5$, classify as **0**

Here,

$$0.769 > 0.5$$

So the predicted class is: $Y = 1$

Final Answer

The probability that the customer will purchase the term deposit is: 0.769 or 76.9%

Hence, the customer is classified as: *Will Purchase the Tem Deposit*

The interpretation of the results obtained from above numerical reflects that the Logistic Regression model predicts that the customer has approximately **76.9% probability** of purchasing the term deposit. Since this probability is greater than the threshold value of 0.5, the customer is placed in the “purchase” category. This means that based on the trained model, a customer with a monthly income of ₹90,000 is more likely to purchase the term deposit than not purchase it.

In practical terms, this probability-based result is useful because it does not only give a final class label, but also shows the strength of prediction. A value of 76.9% indicates fairly strong confidence in the positive outcome.

Stepwise summary of the steps performed in above numerical

Given: $z = -6 + 0.08x, x = 90$

Step 1: $z = -6 + 0.08(90) = 1.2$

Step 2: $P(Y = 1 | X) = \frac{1}{1 + e^{-1.2}} = \frac{1}{1 + 0.3010} = \frac{1}{1.3010} \approx 0.769$

Step 3: Since $0.769 > 0.5$

the customer belongs to class 1. i.e. Customer will purchase the Term deposit.

Important Note:

In Logistic Regression numerical, always follow the steps, given below:

- First, calculate the linear expression: $z = \beta_0 + \beta_1 x$
- Second, convert this value into probability using the sigmoid function:

$$P(Y = 1 | X) = \frac{1}{1 + e^{-z}}$$

- Third, compare the obtained probability with the threshold, usually 0.5.
- If the probability is greater than or equal to 0.5, classify the observation into class 1. Otherwise, classify it into class 0.

So, Logistic Regression works in two stages: it first forms a linear score, and then converts that score into a probability for classification.

This type of numerical is commonly used in banking, healthcare, and marketing. For example, instead of income, the feature could be exam score, age, blood pressure, or advertisement clicks.

Logistic Regression helps convert such input values into probabilities and then makes a classification decision.

The advantages and the Limitations of the Logistic Regression are as follows:

Advantages of Logistic Regression

- Simple and easy to understand and implement
- Provides probabilistic output (useful for decision-making)
- Works well for linearly separable data
- Computationally efficient
- Less prone to overfitting when regularization is used

Limitations of Logistic Regression

- Assumes a linear relationship between features and log-odds
- Not suitable for complex non-linear data
- Sensitive to outliers
- Performance decreases when features are highly correlated (multicollinearity)
- Requires feature scaling for better performance

From above discussion we learned that, Logistic Regression can be understood as a bridge between linear models and probability theory. It takes a linear equation and transforms it into a probability using the sigmoid function, making it highly interpretable and practical. The key idea is not just predicting a class but understanding the likelihood of an outcome. In real-world scenarios such as medical diagnosis, credit scoring, and marketing predictions, this probabilistic interpretation makes Logistic Regression a powerful and reliable tool.

☛ Check Your Progress 5:

Q1. What is Logistic Regression? How is it different from Linear Regression?

.....

.....
Q2. Explain the sigmoid (logistic) function used in Logistic Regression.
.....

.....
Q3. How does Logistic Regression perform classification?
.....

.....
Q4. What are the advantages of Logistic Regression?
.....

.....
Q5. What are the limitations of Logistic Regression?
.....

10.3.4 DECISION TREES

Decision Trees are one of the most intuitive and widely used supervised learning algorithms for both classification and regression problems. The core idea of a Decision Tree is to break down a complex decision-making process into a series of simple, sequential decisions, represented in the form of a tree-like structure. Each internal node of the tree represents a test on a feature, each branch represents the outcome of that test, and each leaf node represents the final prediction or class label. This structure closely resembles human decision-making, making Decision Trees easy to understand and interpret.

In a typical Decision Tree, the algorithm starts from the root node and recursively splits the dataset into smaller subsets based on the feature that best separates the data. The goal is to create pure subsets, where most of the data points belong to a single class. To determine the best feature for splitting, the algorithm uses measures such as **Entropy**, **Information Gain**, or **Gini Index**.

The concept of Entropy is used to measure the impurity or randomness in the dataset. It is defined as:

$$Entropy(S) = -\sum p_i \log_2 p_i$$

where p_i represents the probability of each class in the dataset.

If all data points belong to one class, entropy is 0 (pure), whereas if the data is evenly distributed among classes, entropy is maximum (impure).

Information Gain is used to decide which feature should be selected for splitting. It is calculated as:

$$IG(S, A) = Entropy(S) - \sum \frac{|S_v|}{|S|} Entropy(S_v)$$

where S is the dataset, A is the feature, and S_v are the subsets formed after splitting.

Just for example, let's consider a scenario where a bank wants to decide whether to approve a loan based on features such as Income (High/Low) and Credit Score (Good/Poor). A Decision Tree will split the data step by step, for example:

- First split: Credit Score
- If Credit Score = Good → Approve Loan
- If Credit Score = Poor → Check Income

- Income = High → Approve
- Income = Low → Reject

This hierarchical decision-making mimics real-world reasoning and provides clear decision rules.

Example5: Calculate Entropy and Information Gain for Income as a feature in a dataset of 10 customers given below:

Class	Count
Yes (Buy)	6
No (Not Buy)	4

Solution:

Step 1: Calculate Entropy of Dataset

$$Entropy(S) = -\left(\frac{6}{10} \log_2 \frac{6}{10} + \frac{4}{10} \log_2 \frac{4}{10}\right)$$

$$Entropy(S) = -(0.6 \log_2 0.6 + 0.4 \log_2 0.4)$$

$$Entropy(S) = -(0.6 \times -0.737 + 0.4 \times -1.322)$$

$$Entropy(S) = 0.971$$

Step 2: Consider a Feature (e.g., Income)

Suppose:

Income	Yes	No	Total
High	4	1	5
Low	2	3	5

Entropy for High Income

$$Entropy(High) = -\left(\frac{4}{5} \log_2 \frac{4}{5} + \frac{1}{5} \log_2 \frac{1}{5}\right) = 0.722$$

Entropy for Low Income

$$Entropy(Low) = -\left(\frac{2}{5} \log_2 \frac{2}{5} + \frac{3}{5} \log_2 \frac{3}{5}\right) = 0.971$$

Step 3: Calculate Information Gain

$$IG(S, Income) = 0.971 - \left(\frac{5}{10} \times 0.722 + \frac{5}{10} \times 0.971\right)$$

$$IG = 0.971 - (0.361 + 0.485)$$

$$IG = 0.125$$

Step 4: Decision: The feature with the highest Information Gain is selected for splitting. If Income gives the highest gain among all features, it becomes the root node.

It is observed in the above numerical that the Information Gain value of 0.125 indicates that splitting the dataset based on Income reduces uncertainty in the dataset. Although the reduction is not very large, it still improves classification. The Decision Tree algorithm will compare this gain with other features and choose the one that maximizes purity at each step.

Let's understand the Decision Tree Algorithm with one more Comprehensive Numerical citing the use on entropy and information gain only.

Example6: A company wants to predict whether a customer will **Buy a Product** or **Not Buy a Product** based on the following attributes:

- **Age:** Young, Middle, Old
- **Income:** High, Medium, Low
- **Student:** Yes, No

The training dataset is given below:

Record	Age	Income	Student	Buy Product
1	Young	High	No	No
2	Young	High	No	No
3	Middle	High	No	Yes
4	Old	Medium	No	Yes
5	Old	Low	Yes	Yes
6	Old	Low	Yes	No
7	Middle	Low	Yes	Yes
8	Young	Medium	No	No
9	Young	Low	Yes	Yes
10	Old	Medium	Yes	Yes

Using the **Decision Tree algorithm with Entropy and Information Gain**, determine the **best attribute for the root node** and continue the splitting process until the class decision is obtained. Show all calculations step by step and write the final decision rules.

Solution: In this problem, we have to construct a Decision Tree for classification. The target variable is:

$$Y = \text{Buy Product}$$

where:

- **Yes** = Customer buys the product
- **No** = Customer does not buy the product

The Decision Tree chooses the attribute that gives the **highest Information Gain** at each stage.

The used Equations are:

(a) Entropy

Entropy measures the impurity or randomness in the dataset.

$$Entropy(S) = - \sum_{i=1}^n p_i \log_2 p_i$$

For binary classification:

$$Entropy(S) = -p_{yes} \log_2(p_{yes}) - p_{no} \log_2(p_{no})$$

(b) Information Gain

Information Gain tells us how much uncertainty is reduced after splitting on an attribute.

$$IG(S, A) = Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Where,

- S : Entire dataset
- A : Attribute selected for splitting
- S_v : Subset of dataset for a particular value v of attribute A
- p_{yes} : Proportion of Yes class
- p_{no} : Proportion of No class
- $Entropy(S)$: Impurity of dataset S
- $IG(S, A)$: Information gain obtained by splitting on attribute A

To solve the given problem the steps to be performed are as follows

Step 1: Calculate Entropy of the Full Dataset:

From the table: Total records = 10 ; Yes = 6 and No = 4

So,

$$p_{yes} = \frac{6}{10} = 0.6$$

$$p_{no} = \frac{4}{10} = 0.4$$

Now entropy of the whole dataset:

$$Entropy(S) = -0.6 \log_2(0.6) - 0.4 \log_2(0.4)$$

Using:

$$\log_2(0.6) \approx -0.737$$

$$\log_2(0.4) \approx -1.322$$

So,

$$Entropy(S) = -0.6(-0.737) - 0.4(-1.322)$$

$$Entropy(S) = 0.4422 + 0.5288$$

$$Entropy(S) = 0.971$$

This means the dataset is impure and needs splitting.

Step 2: Calculate Information Gain for Attribute = Age

In the Given table, the attribute **Age** has three values: Young; Middle; Old

(a) Subset for Age = Young

Refer to Given Table Records: 1, 2, 8, 9

Refer to Given Table Class labels(Buy Product): No, No, No, Yes

- Yes = 1
- No = 3

$$Entropy(Young) = -\frac{1}{4} \log_2 \frac{1}{4} - \frac{3}{4} \log_2 \frac{3}{4}$$

$$Entropy(Young) = -0.25(-2) - 0.75(-0.415)$$

$$Entropy(Young) = 0.5 + 0.311$$

$$Entropy(Young) = 0.811$$

(b) Subset for Age = Middle

Refer to Given Table Records: 3, 7

Refer to Given Table Class labels(Buy Product): Yes, Yes

- Yes = 2
- No = 0

$$Entropy(Middle) = 0$$

(c) Subset for Age = Old

Refer to Given Table Records: 4, 5, 6, 10

Refer to Given Table Class labels(Buy Product): Yes, Yes, No, Yes

- Yes = 3
- No = 1

$$Entropy(Old) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4}$$

$$Entropy(Old) = -0.75(-0.415) - 0.25(-2)$$

$$Entropy(Old) = 0.311 + 0.5$$

$$Entropy(Old) = 0.811$$

(d) Weighted Entropy for Age

$$Entropy(S, Age) = \frac{4}{10}(0.811) + \frac{2}{10}(0) + \frac{4}{10}(0.811)$$

$$Entropy(S, Age) = 0.3244 + 0 + 0.3244$$

$$Entropy(S, Age) = 0.6488$$

(e) Information Gain for Age

$$IG(S, Age) = 0.971 - 0.6488$$

$$IG(S, Age) = 0.3222$$

Result: $IG(S, Age) = 0.3222$

Step 3: Calculate Information Gain for Attribute = Income

Income values: High, Medium, and Low

(a) Subset for Income = High

Refer to Given Table Records: 1, 2, 3

Refer to Given Table Class labels(Buy Product): No, No, Yes

- Yes = 1
- No = 2

$$Entropy(High) = -\frac{1}{3} \log_2 \frac{1}{3} - \frac{2}{3} \log_2 \frac{2}{3}$$

Using:

$$\log_2(1/3) \approx -1.585, \log_2(2/3) \approx -0.585$$

$$Entropy(High) = -\frac{1}{3}(-1.585) - \frac{2}{3}(-0.585)$$

$$Entropy(High) = 0.528 + 0.390$$

$$Entropy(High) = 0.918$$

(b) Subset for Income = Medium

Refer to Given Table Records: 4, 8, 10

Refer to Given Table Class labels(Buy Product): Yes, No, Yes

- Yes = 2
- No = 1

$$Entropy(Medium) = 0.918$$

(c) Subset for Income = Low

Refer to Given Table Records: 5, 6, 7, 9

Refer to Given Table Class labels(Buy Product): Yes, No, Yes, Yes

- Yes = 3
- No = 1

$$Entropy(Low) = 0.811$$

(d) Weighted Entropy for Income

$$\begin{aligned}
 Entropy(S, Income) &= \frac{3}{10}(0.918) + \frac{3}{10}(0.918) + \frac{4}{10}(0.811) \\
 Entropy(S, Income) &= 0.2754 + 0.2754 + 0.3244 \\
 Entropy(S, Income) &= 0.8752
 \end{aligned}$$

(e) Information Gain for Income

$$IG(S, Income) = 0.971 - 0.8752$$

$$IG(S, Income) = 0.0958$$

$$\text{Result: } IG(S, Income) = 0.0958$$

Step 4: Calculate Information Gain for Attribute = Student

Student values: Yes, No

(a) Subset for Student = Yes

Refer to Given Table Records: 5, 6, 7, 9, 10

Refer to Given Table Class labels(Buy Product): Yes, No, Yes, Yes, Yes

- Yes = 4
- No = 1

$$\begin{aligned}
 Entropy(Yes) &= -\frac{4}{5} \log_2 \frac{4}{5} - \frac{1}{5} \log_2 \frac{1}{5} \\
 Entropy(Yes) &= -0.8(-0.322) - 0.2(-2.322) \\
 Entropy(Yes) &= 0.258 + 0.464 \\
 Entropy(Yes) &= 0.722
 \end{aligned}$$

(b) Subset for Student = No

Refer to Given Table Records: 1, 2, 3, 4, 8

Refer to Given Table Class labels(Buy Product): No, No, Yes, Yes, No

- Yes = 2
- No = 3

$$\begin{aligned}
 Entropy(No) &= -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} \\
 Entropy(No) &= 0.971
 \end{aligned}$$

(c) Weighted Entropy for Student

$$\begin{aligned}
 Entropy(S, Student) &= \frac{5}{10}(0.722) + \frac{5}{10}(0.971) \\
 Entropy(S, Student) &= 0.361 + 0.4855 \\
 Entropy(S, Student) &= 0.8465
 \end{aligned}$$

(d) Information Gain for Student

$$IG(S, Student) = 0.971 - 0.8465$$

$$IG(S, Student) = 0.1245$$

$$\text{Result: } IG(S, Student) = 0.1245$$

Step 5: Select the Root Node

Now compare all Information Gain values:

- $IG(S, Age) = 0.322$
- $IG(S, Income) = 0.096$
- $IG(S, Student) = 0.125$

The highest Information Gain is for **Age**.

Therefore, the root node is: Age

Step 6: Continue Splitting the Branches

We now split according to Age.

Branch 1: Age = Middle

Records: 3, 7

Both are **Yes**

So this is a pure node. Is Age = Middle → Buy Product = Yes

Branch 2: Age = Young

Subset:

Record	Age	Income	Student	Buy Product
1	Young	High	No	No
2	Young	High	No	No
8	Young	Medium	No	No
9	Young	Low	Yes	Yes

Class counts:

- Yes = 1
- No = 3

$$Entropy(Young) = 0.811$$

Now among remaining attributes, test **Income** and **Student**.

(a) Split Young subset by Student

Student = No

Records: 1, 2, 8

Class labels: No, No, No

$$Entropy = 0$$

Student = Yes

Record: 9

Class label: Yes

$$Entropy = 0$$

Weighted entropy:

$$Entropy(Young, Student) = \frac{3}{4}(0) + \frac{1}{4}(0) = 0$$

Information Gain:

$$IG(Young, Student) = 0.811 - 0 = 0.811$$

(b) Split Young subset by Income

Income = High

Records: 1, 2

Both No → Entropy = 0

Income = Medium

Record: 8

No → Entropy = 0

Income = Low

Record: 9

Yes → Entropy = 0

Weighted entropy: $Entropy(Young, Income) = 0$

Information Gain: $IG(Young, Income) = 0.811 - 0 = 0.811$

Both Student and Income perfectly classify this subset. We can choose either. Usually, one may choose **Student** because it gives a simpler rule here.

So:

- Age = Young and Student = No $\rightarrow$ No
- Age = Young and Student = Yes $\rightarrow$ Yes

Branch 3: Age = Old

Subset:

Record Age Income Student Buy Product

4	Old	Medium	No	Yes
5	Old	Low	Yes	Yes
6	Old	Low	Yes	No
10	Old	Medium	Yes	Yes

Class counts:

- Yes = 3
- No = 1

$$Entropy(Old) = 0.811$$

Now consider remaining attributes: **Income** and **Student**

(a) Split Old subset by Income

Income = Medium

Records: 4, 10

Class labels: Yes, Yes

$$Entropy = 0$$

Income = Low

Records: 5, 6

Class labels: Yes, No

$$Entropy = -\frac{1}{2} \log_2 \frac{1}{2} - \frac{1}{2} \log_2 \frac{1}{2} = 1$$

Weighted entropy:

$$Entropy(Old, Income) = \frac{2}{4}(0) + \frac{2}{4}(1) = 0.5$$

Information Gain:

$$IG(Old, Income) = 0.811 - 0.5 = 0.311$$

(b) Split Old subset by Student

Student = Yes

Records: 5, 6, 10

Class labels: Yes, No, Yes

- Yes = 2
- No = 1

$$Entropy = 0.918$$

Student = No

Record: 4

Class label: Yes

$$Entropy = 0$$

Weighted entropy:

$$Entropy(Old, Student) = \frac{3}{4}(0.918) + \frac{1}{4}(0)$$

$$Entropy(Old, Student) = 0.6885$$

Information Gain:

$$IG(Old, Student) = 0.811 - 0.6885 = 0.1225$$

So the better split is **Income**.

Thus:

- Age = Old and Income = Medium → Yes
- Age = Old and Income = Low → still mixed, so split again

Step 7: Final Split for Age = Old and Income = Low

Subset:

Record	Age	Income	Student	Buy Product
5	Old	Low	Yes	Yes
6	Old	Low	Yes	No

Here all remaining records have Student = Yes, so there is no useful further split with the available attributes. In exam problems, this situation is often handled by assigning the **majority class** of the parent node or noting that data is insufficient to split further uniquely. Since in the Old branch overall majority class is **Yes**, we assign:

Age = Old and Income = Low → Buy Product = Yes (majority)

Final Decision Rules

The Decision Tree can be written as rules:

- Rule 1: If Age = Middle, then Buy Product = Yes
- Rule 2: If Age = Young and Student = No, then Buy Product = No
- Rule 3: If Age = Young and Student = Yes, then Buy Product = Yes
- Rule 4: If Age = Old and Income = Medium, then Buy Product = Yes
- Rule 5: If Age = Old and Income = Low, then Buy Product = Yes (majority rule)

Results of this numerical shows that **Age** is the most important attribute because it gives the highest Information Gain and therefore becomes the root node of the Decision Tree. This means Age reduces uncertainty more than Income and Student in this dataset. After selecting Age, the algorithm continues to divide the data into smaller and purer subsets. For the Young group, Student status becomes important, while for the Old group, Income becomes the better splitting factor. The final tree provides clear and interpretable rules for classification.

In practical terms, the Decision Tree is helpful because it converts raw data into simple human-readable decisions such as “If Age is Young and Student is No, then customer will not buy the product.” This is why Decision Trees are widely used in banking, marketing, healthcare, and business decision-making.

While solving Decision Tree numerical, always follow this sequence:

- First, calculate the entropy of the full dataset.
- Second, calculate the entropy of subsets formed by each attribute.
- Third, compute Information Gain for each attribute.
- Fourth, choose the attribute with the highest Information Gain as the root node.

- Fifth, repeat the same process for each non-pure branch until pure nodes or majority-based terminal nodes are obtained.

So, the full logic is: Entropy → Information Gain → Best Split → Repeat

This stepwise process helps in understanding not only how the tree is built but also why a particular attribute is chosen at each stage.

Here is One more Numerical on Decision Tree Using Gini Index

Example 6: A coaching institute wants to predict whether a student will **Pass** or **Fail** based on the following attributes:

- **Attendance:** High, Low
- **Assignment Submission:** Yes, No

The training dataset is given below:

Record	Attendance	Assignment Submission	Result
1	High	Yes	Pass
2	High	Yes	Pass
3	High	No	Pass
4	High	No	Fail
5	Low	Yes	Pass
6	Low	Yes	Fail
7	Low	No	Fail
8	Low	No	Fail

Using the **Decision Tree algorithm with Gini Index**, determine the **best attribute for the root node**. Show all calculations step by step and write the final decision rules.

Solution: Objective of the given problem is to construct a Decision Tree for classification. The target variable is:

$$Y = \text{Result}$$

where:

- *Pass* = student passes the exam
- *Fail* = student fails the exam

In Gini-based Decision Tree, the best split is the one with the **lowest weighted Gini Index**.

Formula Used for the Gini Index of a dataset is: $Gini(S) = 1 - \sum_{i=1}^n p_i^2$

For binary classification: $Gini(S) = 1 - (p_{pass}^2 + p_{fail}^2)$

Weighted Gini after splitting on attribute A is: $Gini(S, A) = \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \times Gini(S_v)$

Where,

- S: complete dataset

- A : attribute chosen for splitting
- S_v : subset of data for a particular value v of attribute A
- p_{pass} : proportion of Pass cases
- p_{fail} : proportion of Fail cases
- $Gini(S)$: impurity of dataset S
- $Gini(S, A)$: weighted impurity after splitting on attribute A

Note: A smaller Gini value means a purer node.

The stepwise solution of the given problem is as follows:

Step 1: Calculate Gini Index of Full Dataset

From the table:

- Total records = 8
- Pass = 4
- Fail = 4

So,

$$p_{pass} = \frac{4}{8} = 0.5$$

$$p_{fail} = \frac{4}{8} = 0.5$$

Now,

$$Gini(S) = 1 - (0.5^2 + 0.5^2)$$

$$Gini(S) = 1 - (0.25 + 0.25)$$

$$Gini(S) = 1 - 0.5 = 0.5$$

Result: $Gini(S) = 0.5$

This shows that the dataset is impure and needs splitting.

Step 2: Calculate Weighted Gini for Attribute = Attendance

Attendance has two values:

- High
- Low

(a) Subset for Attendance = High

Records: 1, 2, 3, 4

Class labels: Pass, Pass, Pass, Fail

- Pass = 3
- Fail = 1

$$p_{pass} = \frac{3}{4}, p_{fail} = \frac{1}{4}$$

$$Gini(High) = 1 - \left(\left(\frac{3}{4} \right)^2 + \left(\frac{1}{4} \right)^2 \right)$$

$$Gini(High) = 1 - \left(\frac{9}{16} + \frac{1}{16} \right)$$

$$Gini(High) = 1 - \frac{10}{16} = \frac{6}{16} = 0.375$$

(b) Subset for Attendance = Low

Records: 5, 6, 7, 8

Class labels: Pass, Fail, Fail, Fail

- Pass = 1
- Fail = 3

$$p_{pass} = \frac{1}{4}, p_{fail} = \frac{3}{4}$$

$$Gini(Low) = 1 - \left(\left(\frac{1}{4} \right)^2 + \left(\frac{3}{4} \right)^2 \right)$$

$$Gini(Low) = 1 - \left(\frac{1}{16} + \frac{9}{16} \right)$$

$$Gini(Low) = 1 - \frac{10}{16} = \frac{6}{16} = 0.375$$

(c) Weighted Gini for Attendance

$$Gini(S, Attendance) = \frac{4}{8} (0.375) + \frac{4}{8} (0.375)$$

$$Gini(S, Attendance) = 0.1875 + 0.1875 = 0.375$$

Result: $Gini(S, Attendance) = 0.375$

Step 3: Calculate Weighted Gini for Attribute = Assignment Submission

Assignment Submission has two values:

- Yes
- No

(a) Subset for Assignment = Yes

Records: 1, 2, 5, 6

Class labels: Pass, Pass, Pass, Fail

- Pass = 3
- Fail = 1

$$Gini(Yes) = 1 - \left(\left(\frac{3}{4} \right)^2 + \left(\frac{1}{4} \right)^2 \right)$$

$$Gini(Yes) = 1 - \left(\frac{9}{16} + \frac{1}{16} \right) = 0.375$$

(b) Subset for Assignment = No

Records: 3, 4, 7, 8

Class labels: Pass, Fail, Fail, Fail

- Pass = 1
- Fail = 3

$$Gini(No) = 1 - \left(\left(\frac{1}{4} \right)^2 + \left(\frac{3}{4} \right)^2 \right)$$

$$Gini(No) = 1 - \left(\frac{1}{16} + \frac{9}{16} \right) = 0.375$$

(c) Weighted Gini for Assignment Submission

$$Gini(S, Assignment) = \frac{4}{8} (0.375) + \frac{4}{8} (0.375)$$

$$Gini(S, Assignment) = 0.375$$

Result: $Gini(S, Assignment) = 0.375$

Step 4: Compare the Split Values

- $Gini(S, \text{Attendance}) = 0.375$
- $Gini(S, \text{Assignment}) = 0.375$

Both attributes give the same weighted Gini Index.

Decision: Since both splits are equally good, we may choose either attribute as the root node. Let us choose: Attendance as the root node.

Step 5: Continue Splitting the Branches:

Now split the data based on **Attendance**.

Branch 1: Attendance = High

Subset:

Record	Attendance	Assignment Submission	Result
1	High	Yes	Pass
2	High	Yes	Pass
3	High	No	Pass
4	High	No	Fail

Class counts:

- Pass = 3
- Fail = 1

$$Gini(High) = 0.375$$

Now only one attribute remains: **Assignment Submission**

Split this subset by Assignment Submission

Assignment = Yes

Records: 1, 2

Class labels: Pass, Pass

$$Gini = 1 - (1^2 + 0^2) = 0$$

Assignment = No

Records: 3, 4

Class labels: Pass, Fail

$$Gini = 1 - \left(\left(\frac{1}{2} \right)^2 + \left(\frac{1}{2} \right)^2 \right)$$

$$Gini = 1 - (0.25 + 0.25) = 0.5$$

Weighted Gini for this split

$$Gini(High, \text{Assignment}) = \frac{2}{4}(0) + \frac{2}{4}(0.5)$$

$$Gini(High, \text{Assignment}) = 0 + 0.25 = 0.25$$

So, this split improves purity.

Thus:

- Attendance = High and Assignment = Yes $\rightarrow$ Pass
- Attendance = High and Assignment = No $\rightarrow$ mixed node

Since no further attribute remains, we assign the **majority class** of this node. In this node, one Pass and one Fail are present, so tie handling is needed. In exam settings, we usually use the parent majority class. Parent node majority = Pass.

So,

- Attendance = High and Assignment = No → Pass

Branch 2: Attendance = Low

Subset:

Record Attendance Assignment Submission Result

5	Low	Yes	Pass
6	Low	Yes	Fail
7	Low	No	Fail
8	Low	No	Fail

Class counts:

- Pass = 1
- Fail = 3

$$Gini(Low) = 0.375$$

Now split by **Assignment Submission**

Assignment = Yes

Records: 5, 6

Class labels: Pass, Fail

$$Gini = 1 - \left(\left(\frac{1}{2} \right)^2 + \left(\frac{1}{2} \right)^2 \right) = 0.5$$

Assignment = No

Records: 7, 8

Class labels: Fail, Fail

$$Gini = 1 - (0^2 + 1^2) = 0$$

Weighted Gini for this split

$$Gini(Low, Assignment) = \frac{2}{4}(0.5) + \frac{2}{4}(0)$$

$$Gini(Low, Assignment) = 0.25$$

Thus:

- Attendance = Low and Assignment = No → Fail
- Attendance = Low and Assignment = Yes → mixed node

Again, no further attribute is available, so use majority class of parent node. Parent node majority = Fail.

So,

- Attendance = Low and Assignment = Yes → Fail

Final Decision Tree

Root node: **Attendance**

High → check **Assignment Submission**

- Yes → Pass
- No → Pass (majority rule)

Low → check **Assignment Submission**

- Yes → Fail (majority rule)
- No → Fail

Final Decision Rules

- *Rule 1: If Attendance = High and Assignment Submission = Yes, then Result = Pass*
- *Rule 2: If Attendance = High and Assignment Submission = No, then Result = Pass (by majority rule)*
- *Rule 3: If Attendance = Low and Assignment Submission = Yes, then Result = Fail (by majority rule)*
- *Rule 4: If Attendance = Low and Assignment Submission = No, then Result = Fail*

Results of this numerical shows how the Gini Index is used to build a Decision Tree by selecting the split with the lowest impurity. Here, both attributes, Attendance and Assignment Submission, produced the same weighted Gini value of 0.375 at the root level. Therefore, either attribute could be chosen as the root node. After selecting Attendance, the remaining attribute helped reduce impurity further within each branch.

The final rules show a simple interpretation: students with high attendance are generally predicted to pass, while students with low attendance are generally predicted to fail. This is a practical and human-readable outcome, which is one of the biggest advantages of Decision Trees.

While solving Gini-based Decision Tree numerical, follow this sequence:

- First, calculate the Gini Index of the full dataset.
- Second, split the dataset by each attribute and calculate the weighted Gini after each split.
- Third, choose the attribute with the smallest weighted Gini as the best split.
- Fourth, repeat the process for each branch until pure or nearly pure nodes are obtained.
- If no further attribute is left and the node is still mixed, apply the majority class rule.

So, the logic is: Gini of dataset → Weighted Gini of each split → Best split → Repeat

The advantages and disadvantages of the decision trees are as follows:

Advantages of Decision Trees

- Decision Trees are easy to understand and interpret, making them highly suitable for decision-making systems.
- They require minimal data preprocessing and can handle both numerical and categorical data.
- They are also capable of capturing non-linear relationships and interactions between features. Additionally, they provide clear decision rules that can be visualized and explained.

Limitations of Decision Trees

- Decision Trees are prone to overfitting, especially when the tree becomes too deep.
- They can be sensitive to small changes in data, which may lead to different tree structures.
- They may also become biased if the dataset is imbalanced.

- Furthermore, single Decision Trees often do not provide the best predictive performance compared to ensemble methods such as Random Forests.

This section concludes with the understanding that the Decision Trees can be understood as a systematic way of asking questions to arrive at a decision. At each step, the algorithm selects the best question (feature) that reduces uncertainty the most. This process continues until the data is sufficiently pure. The use of entropy and information gain provides a mathematical foundation for what appears to be a simple decision-making process. Because of their interpretability and logical structure, Decision Trees form the basis for many advanced algorithms and are an essential concept in machine learning.

☛ Check Your Progress 5 6:

Q1. What is a Decision Tree? Explain its basic structure.

.....

.....

Q2. How does a Decision Tree work in classification problems?

.....

.....

Q3. What are the advantages of Decision Trees?

.....

.....

Q4. What are the limitations of Decision Trees?

.....

.....

Q5. What is overfitting in Decision Trees and how can it be controlled?

.....

.....

10.3.5 RANDOM FOREST

Decision Trees and Random Forest are both supervised learning algorithms based on tree structures, but they differ significantly in their approach and performance.

A Decision Tree is a single model that splits the dataset into smaller subsets based on feature conditions, forming a tree-like structure of decisions. It is simple, easy to understand, and highly interpretable, as the decision-making process can be clearly visualized through rules. However, this simplicity comes with a drawback: Decision Trees are highly prone to overfitting, especially when the tree becomes deep and captures noise in the data. They are also sensitive to small changes in the dataset, which can lead to completely different tree structures.

On the other hand, Random Forest is an ensemble learning technique that builds multiple Decision Trees and combines their predictions to produce a final result. Instead of relying on a single tree, it uses techniques such as bootstrapping (random sampling of data) and random feature selection to create diversity among trees. The final prediction is obtained through majority voting in classification or averaging in regression. This approach significantly improves accuracy and reduces overfitting, making Random Forest more robust and reliable compared to a single Decision Tree.

In terms of performance, Random Forest generally outperforms Decision Trees because it reduces variance and improves generalization. However, this comes at the cost of increased complexity and reduced interpretability. While a Decision Tree provides clear and straightforward decision rules, Random Forest behaves more like a black-box model where individual decisions are harder to interpret. Additionally, Random Forest requires more computational resources and training time due to the construction of multiple trees, whereas Decision Trees are faster and more efficient for smaller datasets.

It is to be noted that the Decision Trees are best suited for problems where interpretability and simplicity are important, while Random Forest is preferred when higher accuracy, robustness, and better generalization are required.

Now, we Understand the Random Forest in more detail,

Random Forest Algorithm is an advanced supervised learning algorithm that improves the performance of Decision Trees by combining multiple trees to make better predictions. It is based on the concept of **ensemble learning**, where instead of relying on a single model, multiple models are trained and their outputs are combined.

The main idea behind Random Forest is that a group of weak learners (decision trees) can come together to form a strong learner. Each tree in the forest is trained on a random subset of the data and uses a random subset of features. This randomness helps in reducing overfitting and improves generalization.

Working of Random Forest: The Random Forest algorithm works in a step-by-step manner to improve prediction accuracy by combining multiple decision trees.

- First, it creates several new datasets from the original dataset using a method called **Bootstrapping (Sampling with replacement)**, where data points are randomly selected with replacement. This means some data points may appear multiple times while others may not appear at all in a particular sample.
- Next, for each of these datasets, a separate **Decision Tree** is built. So instead of having one tree, we now have multiple trees trained on slightly different data.
- While constructing each tree, the algorithm does not consider all features at every split. Instead, it selects a **random subset of features** for splitting. This introduces diversity among the trees and helps reduce overfitting.
- Finally, once all the trees are built, their predictions are combined to produce the final output.
 - In case of classification problems, the final prediction is made using **majority voting**, meaning the class predicted by most trees is selected.
 - In case of regression problems, the final output is calculated by taking the **average of predictions** from all trees.

In this way, Random Forest improves accuracy and stability by combining the results of multiple decision trees instead of relying on just one.

So far as the Mathematical Representation of Random Forest is concerned, in any random Forest the final prediction is obtained by combining the outputs of multiple decision trees.

- For **classification problems**, each tree gives a class prediction, and the final result is decided based on **majority voting**. This means the class that is predicted by the highest number of trees is selected as the final output.

$\hat{Y}$ = Majority vote of all trees

For **regression problems**, each tree gives a numerical value, and the final prediction is calculated by taking the **average of all tree outputs**.

$$\hat{Y} = \frac{1}{N} \sum_{i=1}^N T_i(x)$$

Where:

- N : total number of decision trees in the forest
- $T_i(x)$: prediction made by the i^{th} decision tree for input x
- $\hat{Y}$: final predicted output

In simple terms, Random Forest combines multiple predictions to produce a more accurate and stable result.

Just for example consider a medical diagnosis system where multiple doctors (trees) give their opinion on whether a patient has a disease or not. Instead of trusting one doctor, we take the majority opinion of all doctors. If most doctors say “Yes,” the final decision is “Yes.” This reduces the chances of wrong diagnosis, similar to how Random Forest works.

In terms of the numerical example suppose a Random Forest model consists of **5 decision trees** predicting whether a student will pass or fail.

Predictions of trees:

Tree	Prediction
T1	Pass
T2	Fail
T3	Pass
T4	Pass
T5	Fail

Step 1: Count Votes

- Pass = 3
- Fail = 2

Step 2: Apply Majority Voting: $\hat{Y}$ = Pass

Final Answer is that the student will Pass

The Results of the above numerical can be interpreted as, The Random Forest model predicts that the student will pass because the majority of trees (3 out of 5) voted for the “Pass” class. This demonstrates how combining multiple models reduces the impact of individual errors and leads to more reliable predictions.

Let’s Understand the Random Forest Algorithm with one more Comprehensive Numerical

Example6: A college wants to predict whether a student will **Pass** or **Fail** in the final examination using the **Random Forest algorithm**. After training the Random Forest model, 5 different decision trees are generated. For a new student, the predictions given by these trees are as follows:

Tree No.	Prediction
Tree 1	Pass
Tree 2	Fail

Tree No.	Prediction
Tree 3	Pass
Tree 4	Pass
Tree 5	Fail

For the same student, the probability estimates for the class **Pass** given by the 5 trees are:

Tree No.	Probability of Pass
Tree 1	0.80
Tree 2	0.35
Tree 3	0.75
Tree 4	0.90
Tree 5	0.40

Using the Random Forest algorithm, answer the following:

1. Find the final predicted class using **majority voting**.
2. Find the average probability of the class **Pass**.
3. Interpret the result.
4. Write the final conclusion.

Solution:

In the given problem, we have to understand how Random Forest makes a final prediction by combining the outputs of multiple decision trees. Since Random Forest is an **ensemble learning method**, it does not depend on a single tree. Instead, it takes the predictions of many trees and combines them.

For **classification problems**, the final prediction is usually based on:

- **Majority voting**, and sometimes
- **Average probability** across trees

Formula Used in Random Forest are:

(a) Majority Voting for Classification

$$\hat{Y} = \text{Majority vote of all trees}$$

This means the class predicted by the maximum number of trees becomes the final output.

(b) Average Probability of a Class

$$P(\text{Pass}) = \frac{1}{N} \sum_{i=1}^N P_i(\text{Pass})$$

Where:

- N = total number of trees
- $P_i(\text{Pass})$ = probability of class Pass predicted by the i^{th} tree
- $P(\text{Pass})$ = final average probability of Pass

If needed, then:

$$P(\text{Fail}) = 1 - P(\text{Pass})$$

For **regression problems**, each tree gives a numerical value, and the final prediction is calculated by taking the **average of all tree outputs**.

$$\hat{Y} = \frac{1}{N} \sum_{i=1}^N T_i(x)$$

Where,

- $\hat{Y}$: final predicted class
- N : total number of trees in the forest
- T_i : prediction of the i^{th} decision tree
- $P_i(\text{Pass})$: probability assigned by the i^{th} tree for class Pass
- $P(\text{Pass})$: final average probability that the student will pass

Step wise Solution to the given problem is as follows:

Step 1: Write the Tree Predictions

From the question, the tree predictions are:

- Tree 1 $\rightarrow$ Pass
- Tree 2 $\rightarrow$ Fail
- Tree 3 $\rightarrow$ Pass
- Tree 4 $\rightarrow$ Pass
- Tree 5 $\rightarrow$ Fail

Now count the votes.

Count of Pass:3

Count of Fail:2

Step 2: Apply Majority Voting

The formula is:

$$\hat{Y} = \text{Majority vote of all trees}$$

Since the number of Pass votes is greater than the number of Fail votes:

$$3 > 2$$

So, the final predicted class is: $\hat{Y} = \text{Pass}$

Step 3: Compute Average Probability of Pass

The probabilities given by the trees are:

- Tree 1: 0.80
- Tree 2: 0.35
- Tree 3: 0.75
- Tree 4: 0.90
- Tree 5: 0.40

Using the formula:

$$P(\text{Pass}) = \frac{1}{N} \sum_{i=1}^N P_i(\text{Pass})$$

Here, $N = 5$

So,

$$P(\text{Pass}) = \frac{0.80 + 0.35 + 0.75 + 0.90 + 0.40}{5}$$

First add all probabilities:

$$0.80 + 0.35 = 1.15$$

$$1.15 + 0.75 = 1.90$$

$$1.90 + 0.90 = 2.80$$

$$2.80 + 0.40 = 3.20$$

Now divide by 5:

$$P(\text{Pass}) = \frac{3.20}{5}$$

$$P(\text{Pass}) = 0.64$$

Thus, the average probability of Pass is: 0.64 or 64%

Step 4: Compute Probability of Fail

If required, probability of Fail can be found as:

$$P(\text{Fail}) = 1 - P(\text{Pass})$$

$$P(\text{Fail}) = 1 - 0.64$$

$$P(\text{Fail}) = 0.36 \text{ or } 36\%$$

Final Answer

Final predicted class: Pass

Average probability of Pass: 0.64 or 64%

Average probability of Fail: 0.36 or 36%

The Results of the Random Forest model in the above numerical, predicts that the student will **Pass** because the majority of trees, that is, 3 out of 5 trees, voted for the class Pass. This is the first basis of the final decision.

The average probability of Pass is **0.64**, which means the model estimates a **64% chance** that the student will pass the exam. Since this value is greater than 0.5, it supports the majority voting result. Therefore, both the class votes and the average probability indicate that the student is more likely to pass than fail.

This also shows one of the biggest strengths of Random Forest i.e. even if some trees make wrong predictions, the combined decision of many trees often leads to a more reliable final result.

To solve a Random Forest numerical, always follow these steps:

- First, list the prediction given by each tree.
- Second, count how many trees voted for each class.
- Third, choose the class with the highest number of votes as the final predicted class.
- Fourth, if probabilities are given, add the probabilities of the required class from all trees and divide by the total number of trees.
- Fifth, interpret both the final class and the probability.

So, the main idea is: Many tree predictions → Combine them → Final decision

Note: Random Forest works on the principle that a group decision is often better than a single decision.

Advantages and Limitation of Random Forests are as follows:

Advantages of Random Forest

- Random Forest provides high accuracy and performs well on large datasets.
- It reduces overfitting by averaging multiple trees.
- It can handle both classification and regression problems.
- It works well even when there are missing values or noisy data.
- It also provides feature importance, which helps in understanding which variables are most influential.

Limitations of Random Forest

- Random Forest models are more complex and less interpretable compared to Decision Trees.
- They require more computational power and memory.
- Training time is higher due to multiple trees.
- It may not be suitable for real-time systems where quick predictions are required.
- Also, the model can become unnecessarily large if too many trees are used.

We learned in this section that Random Forest can be understood as an improved version of Decision Trees. While a single Decision Tree may give unstable or biased results, Random Forest overcomes this by combining many trees and using randomness. The key idea is that diversity among trees leads to better performance. This makes Random Forest one of the most powerful and widely used algorithms in machine learning.

Check Your Progress 7:

Q1. What is Random Forest? Explain its basic concept..

.....
.....

Q2. How does Random Forest improve over Decision Trees?

.....
.....

Q3. Explain the working of Random Forest..

.....
.....

Q4. What are the advantages of Random Forest?.

.....
.....

Q5. What are the limitations of Random Forest?

.....
.....

10.3.6 SUPPORT VECTOR MACHINES (SVM)

Support Vector Machines (SVM) is a powerful supervised learning algorithm used for both classification and regression problems, although it is most commonly applied to classification tasks. The main objective of SVM is to find the optimal boundary (called a hyperplane) that separates different classes of data in such a way that the margin between them is maximized. Unlike many other algorithms that simply try to separate classes, SVM focuses on finding the best possible separation with the maximum distance from the nearest data points of each class.

Before going into the details of SVM, let's understand the Concept of Hyperplane and Margin: In SVM, a hyperplane is a decision boundary that separates data points belonging to different classes. In a two-dimensional space, this hyperplane is simply a straight line. The key idea is not just to separate the classes, but to maximize the margin, which is the distance between the hyperplane and the closest data points from each class. These closest points are called support vectors, and they play a crucial role in defining the position of the hyperplane.

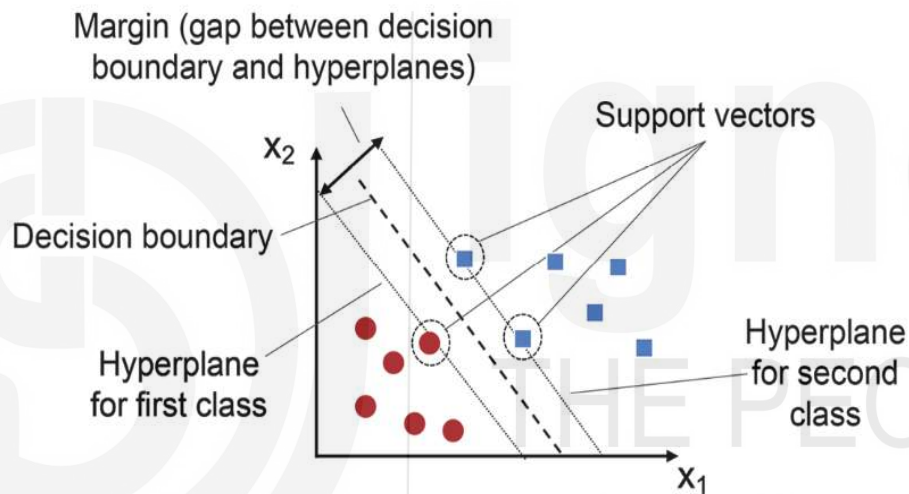


Figure-2 Support Vector Machines (SVM)

The diagram shown above in Figure-2, shows two classes of data points (for example, positive and negative). Multiple lines could separate these classes, but SVM selects the one that maximizes the margin. The points closest to the boundary are the **support vectors**, and the distance between these points and the hyperplane defines the margin. A larger margin generally leads to better generalization and improved model performance.

In Support Vector Machines, the **support vectors** are the most important data points in the dataset because they lie closest to the separating boundary, known as the hyperplane. These points essentially “support” or define the position of the hyperplane. Unlike other points that may lie far away from the boundary, support vectors directly influence how the decision boundary is placed. Even a small change in these points can shift the hyperplane, which shows their critical role in the model.

The concept of **margin** is closely related to support vectors. The margin refers to the distance between the hyperplane and the nearest data points from both classes, which are the support vectors. In SVM, the goal is to maximize this margin so that the separation between classes

becomes as wide as possible, leading to better generalization and reduced chances of misclassification.

Mathematically, the margin is calculated using the formula:

$$\text{Margin} = \frac{2}{\|w\|}$$

where $\|w\|$ represents the magnitude (or length) of the weight vector w . The magnitude of w is computed as:

$$\|w\| = \sqrt{w_1^2 + w_2^2}$$

Here, w_1 and w_2 are the components of the weight vector corresponding to the input features. The value of $\|w\|$ determines how steep or tilted the hyperplane is.

From the formula, we can observe that the margin is inversely proportional to the magnitude of w . This means that if $\|w\|$ is small, the margin becomes larger, and if $\|w\|$ is large, the margin becomes smaller. Therefore, SVM tries to find a hyperplane that minimizes $\|w\|$ in order to maximize the margin.

In simple terms, support vectors are the boundary points that define the decision line, and the margin is the safe distance maintained from these points. A larger margin ensures a more reliable and robust classification model.

The Mathematical Representation of SVM is given by the equation of the hyperplane is given by:

$$w \cdot x + b = 0$$

Where:

- w : weight vector (normal to the hyperplane)
- x : input feature vector
- b : bias term

For classification:

$$\begin{aligned} w \cdot x + b &\geq +1 (\text{Class 1}) \\ w \cdot x + b &\leq -1 (\text{Class 2}) \end{aligned}$$

The margin is given by:

$$\text{Margin} = \frac{2}{\|w\|}$$

SVM aims to maximize this margin.

Just for example consider an email classification system that separates emails into **Spam** and **Not Spam** based on features such as the number of suspicious words and links. SVM will draw a boundary that best separates spam emails from normal emails while ensuring that the closest emails to the boundary are as far apart as possible. This reduces the chances of misclassification.

Let's understand SVM with a simple Numerical Example, given below:

Example 7: Consider the following two data points from two classes:

- Class +1: (2, 2)
- Class -1: (4, 4)

Assume a simple hyperplane:

$$w = [1, -1], \quad b = 0$$

Solution:

Step 1: Check classification for point (2,2)

$$w \cdot x + b = (1)(2) + (-1)(2) + 0 = 2 - 2 = 0$$

Step 2: Check classification for point (4,4)

$$w \cdot x + b = (1)(4) + (-1)(4) = 0$$

Both points lie on the boundary, so this hyperplane is not optimal.

Now consider a better hyperplane:

$$w = [1, -1], \quad b = -1$$

Step 3: Recalculate

- For (2,2): $(1)(2) + (-1)(2) - 1 = -1 \rightarrow$ Class -1
- For (4,4): $(1)(4) + (-1)(4) - 1 = -1 \rightarrow$ Class -1 again (incorrect)

This shows that choosing the correct hyperplane is critical, and SVM mathematically optimizes this selection.

The numerical example highlights that not every separating line is suitable. SVM carefully selects the hyperplane that maximizes the margin while correctly classifying the data. The support vectors define this boundary, and any incorrect choice of parameters leads to misclassification.

Let's understand SVM in more detail through a Numerical given below:

Example 8: Consider the following two classes of data points for a binary classification problem:

- Class +1: $A(3,1), B(3,2)$
- Class -1: $C(1,3), D(2,3)$

Assume that the SVM classifier gives the separating hyperplane:

$$x_1 + x_2 - 4 = 0$$

Using this hyperplane, answer the following:

1. Verify whether the given points are correctly classified.
2. Identify the support vectors.
3. Find the margin of the classifier.
4. Classify a new point $P(4,1)$.
5. Interpret the result.

Solve the problem step by step using the SVM equations, explain all variables used, and present the answer in a self-learning format.

Solution:

We learned that Support Vector Machine is a supervised learning algorithm used mainly for classification. Its goal is to find the best separating boundary, called a hyperplane, between two classes. This hyperplane should separate the classes in such a way that the margin, or the distance between the closest points of the two classes and the hyperplane, is maximum.

In two dimensions, the hyperplane is simply a straight line.

The Equation Used in SVM are as follows:

- The general equation of a hyperplane is: $w \cdot x + b = 0$
- For two features, this becomes: $w_1 x_1 + w_2 x_2 + b = 0$
- For the given problem: $x_1 + x_2 - 4 = 0$
- So, by comparison: $w = (1,1), b = -4$

Where,

- $x = (x_1, x_2)$: input feature vector
- x_1, x_2 : coordinates of a data point
- $w = (w_1, w_2)$: weight vector, perpendicular to the hyperplane
- b : bias or intercept term
- $w \cdot x + b$: decision function
- If $w \cdot x + b > 0$, point belongs to Class +1
- If $w \cdot x + b < 0$, point belongs to Class -1
- Support vectors are the points closest to the hyperplane with Margin given by:

$$\text{Margin} = \frac{2}{\|w\|} \text{ where } \|w\| = \sqrt{w_1^2 + w_2^2}$$

Stepwise solution to the given problem is as follows:

Step 1: Verify Classification of Given Points

We use the decision function:

$$f(x) = x_1 + x_2 - 4$$

Now calculate for each point.

Point A(3,1), Class +1

$$f(A) = 3 + 1 - 4 = 0$$

So point A lies exactly on the boundary.

Point B(3,2), Class +1

$$f(B) = 3 + 2 - 4 = 1$$

Since $f(B) > 0$, it belongs to Class +1.
Correctly classified.

Point C(1,3), Class -1

$$f(C) = 1 + 3 - 4 = 0$$

So, point C also lies exactly on the boundary.

Point D(2,3), Class -1

$$f(D) = 2 + 3 - 4 = 1$$

Since $f(D) > 0$, this point is classified as Class +1.

But actual class is Class -1. So, point D is misclassified.

Step 2: Check Whether the Given Hyperplane is a Proper SVM Separator

From the above calculations:

- A(3,1) lies on boundary

- $C(1,3)$ lies on boundary
- $D(2,3)$ is wrongly classified

Therefore, this hyperplane is not a correct separating hyperplane for all training points.

However, from an exam point of view, we can still use the given line to demonstrate how SVM calculations are done.

Step 3: Identify Support Vectors

Support vectors are the points nearest to the separating hyperplane. Usually, these satisfy:

$$w \cdot x + b = +1 \text{ or } w \cdot x + b = -1$$

But in this simplified question, the closest points are those lying on or nearest to the boundary.

From the calculations:

- $A(3,1): f(A) = 0$
- $C(1,3): f(C) = 0$

These are the nearest points to the hyperplane.

So, the support vectors are: $A(3,1)$ and $C(1,3)$

Step 4: Calculate the Margin

The margin formula is:

$$\text{Margin} = \frac{2}{\|w\|}$$

First calculate $\|w\|$:

$$\|w\| = \sqrt{1^2 + 1^2}$$

$$\|w\| = \sqrt{2}$$

Now substitute:

$$\text{Margin} = \frac{2}{\sqrt{2}}$$

$$\text{Margin} = \sqrt{2}$$

So, the margin is: $\sqrt{2} \approx 1.414$

Step 5: Classify the New Point $P(4,1)$

Use the decision function:

$$f(P) = 4 + 1 - 4$$

$$f(P) = 1$$

Since $f(P) > 0$, the point belongs to: Class +1

Thus, the Final Answer has Support vectors: $A(3,1)$, $C(1,3)$ and Margin: $\sqrt{2} \approx 1.414$
Classification of new point $P(4,1)$: Class +1

This numerical shows how SVM uses the hyperplane equation to classify points. By substituting the coordinates of each point into the decision function, we can determine on which side of the boundary the point lies. The sign of the result tells the predicted class. The

support vectors are the points closest to the boundary, and the margin tells us how far apart the two classes are separated by the hyperplane.

For the new point $P(4,1)$, the value of the decision function is positive, so it is placed in Class +1. This means the point lies on the positive side of the hyperplane. A larger margin generally indicates better class separation and better generalization.

At the same time, this example also teaches an important lesson: not every given line is a perfect SVM separator. Since one of the original training points is misclassified, the given hyperplane is not the ideal one for the whole dataset. In actual SVM training, the algorithm finds the hyperplane that separates the training data as correctly as possible while maximizing the margin.

While solving SVM numericals, always follow this order:

- First, write the hyperplane equation: $w \cdot x + b = 0$
- Second, identify the values of w and b .
- Third, substitute each data point into: $f(x) = w \cdot x + b$
- Fourth, check the sign of the result:
 - Positive $\rightarrow$ Class +1
 - Negative $\rightarrow$ Class -1
 - Zero $\rightarrow$ Point lies on the boundary
- Fifth, find the support vectors, which are the closest points to the boundary.
- Sixth, calculate the margin using: $\text{Margin} = \frac{2}{\|w\|}$
- Finally, classify any new point using the same decision function.

So, the full logic is:

Hyperplane $\rightarrow$ Decision function $\rightarrow$ Class prediction $\rightarrow$ Support vectors $\rightarrow$ Margin

Before concluding this section on SVM, one more important concept needs attention, and that concept is **Kernel Trick**: In many real-world classification problems, the data is not linearly separable, meaning we cannot draw a straight line (or hyperplane) that perfectly separates the classes. In such cases, a simple SVM with a linear boundary fails to classify the data correctly.

To overcome this limitation, SVM uses a powerful concept known as the Kernel Trick. The idea behind the kernel trick is to transform the original data into a higher-dimensional space where the data becomes linearly separable. Instead of explicitly computing this transformation (which can be computationally expensive), SVM uses a mathematical function called a kernel function to directly compute the relationships in the higher-dimensional space.

In simple terms, the kernel trick allows us to solve complex, non-linear problems by converting them into linear problems in a higher dimension—without actually performing the transformation explicitly.

Let's now talk about an example and imagine that data points are arranged in a circular pattern where one class is inside the circle and another is outside. In two dimensions, no straight line can separate these classes. However, if we project this data into a higher

dimension (for example, adding a new feature like radius), the data can now be separated using a straight line (or plane). This transformation is what the kernel trick achieves.

There are several types of kernel, few of the Common Types of Kernels are as follows:

1. Linear Kernel: The linear kernel is the simplest form and is used when the data is already linearly separable.

$$K(x_i, x_j) = x_i \cdot x_j$$

It computes the dot product between two data points. This kernel does not transform the data and works well for simple datasets.

Example: Classifying emails based on word frequency when the data is already separable using a straight line.

2. Polynomial Kernel: The polynomial kernel allows the algorithm to model more complex relationships by considering polynomial combinations of features.

$$K(x_i, x_j) = (x_i \cdot x_j + 1)^d$$

Where d is the degree of the polynomial.

Example: In a student performance dataset, where marks and study hours have a non-linear relationship, a polynomial kernel can capture curved decision boundaries.

3. Radial Basis Function (RBF) Kernel: The RBF kernel (also called Gaussian kernel) is one of the most widely used kernels. It maps data into an infinite-dimensional space and can handle highly non-linear relationships.

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2)$$

Where:

- γ controls how far the influence of a point extends

Example: Image classification or pattern recognition where the data has complex, irregular boundaries.

It is learned that the kernel trick allows SVM to create non-linear decision boundaries while still using linear separation logic internally. Instead of drawing a straight line in the original space, SVM effectively draws a complex curve or boundary by operating in a higher-dimensional space.

It is the kernel trick that makes SVM a powerful and flexible algorithm. It enables SVM to handle both simple linear problems and highly complex non-linear problems without increasing computational complexity significantly. By choosing the appropriate kernel function, SVM can adapt to different types of data and provide accurate classification results.

The Advantages and the Limitations of Support Vector Machines are as follows:

Advantages of SVM:

- SVM is highly effective in high-dimensional spaces and works well when the number of features is large.
- It provides good generalization by maximizing the margin.
- It is robust to overfitting, especially in cases where the margin is large.
- It also works well with clear separation between classes.

Limitations of SVM

- SVM can be computationally expensive for large datasets.
- It is less interpretable compared to Decision Trees.
- Choosing the correct kernel and parameters can be challenging.
- It also performs poorly when there is a lot of noise or overlapping classes in the data.

From the above discussion it is understood that Support Vector Machines combine geometric intuition with mathematical optimization to provide a robust classification method. By focusing on margin maximization and using kernel functions for complex data, SVM becomes a versatile tool in machine learning. It is widely used in applications such as image classification, text categorization, and bioinformatics, where accurate classification is essential.

Actually, SVM is an algorithm that tries to draw the **best possible boundary** between classes, not just any boundary. The idea of maximizing the margin makes it different from other classifiers. Instead of focusing on all data points, SVM focuses only on the most critical ones—the support vectors. This makes it both efficient and powerful.

☛ Check Your Progress 8:

Q1. What is a Support Vector Machine (SVM)? Explain its objective..

.....

Q2. What is a hyperplane and margin in SVM?

.....

Q3. What are Support Vectors? Why are they important?

.....

Q4. How does SVM handle non-linear data?

.....

Q5. What are the advantages and limitations of SVM?

.....

10.4 SUMMARY

This unit provides a comprehensive understanding of machine learning classification techniques, beginning with the fundamental distinction between Supervised and Unsupervised Learning. Supervised learning involves training a model using labelled data, where both input and output are known, enabling the model to learn patterns and make predictions. In contrast, unsupervised learning deals with unlabelled data and focuses on discovering hidden structures or patterns without predefined outputs. This distinction forms the foundation for understanding various supervised learning models discussed in the unit.

The unit then explores several important supervised learning algorithms. It begins with K-Nearest Neighbour (KNN), a simple instance-based algorithm that classifies data points based on the majority class of their nearest neighbors. Next, Naïve Bayes is introduced as a probabilistic classifier based on Bayes' theorem, which assumes independence among features and is widely used in text classification. Logistic Regression is discussed as a statistical method that predicts probabilities using the sigmoid function, making it suitable for binary classification problems.

Further, the unit covers Decision Trees, which model decision-making in a tree-like structure using concepts such as entropy and information gain. While Decision Trees are easy to interpret, they may suffer from overfitting. To address this limitation, the unit introduces Random Forest, an ensemble technique that combines multiple decision trees to improve accuracy and robustness by reducing variance through techniques like bootstrapping and feature randomness.

Finally, the unit focuses on Support Vector Machines (SVM), which aim to find the optimal hyperplane that maximizes the margin between different classes. The concept of support vectors and margin maximization is emphasized as the key strength of SVM. The unit also highlights the importance of the kernel trick, which enables SVM to handle non-linear data by transforming it into higher-dimensional space.

Overall, this unit builds a strong conceptual and practical understanding of major supervised learning algorithms, comparing their working principles, strengths, and limitations. It equips learners with the ability to select appropriate models for different types of classification problems and understand the mathematical intuition behind them.

10.5 ANSWERS

☛ Check Your Progress 1

Q1 What is the main difference between supervised and unsupervised learning?

Solution: Refer to Section 10.2

Q2 Which type of learning is used for predicting house prices and why?

Solution: Refer to Section 10.2

Q3 Give one real-life example of unsupervised learning.

Solution: Refer to Section 10.2

Q4 Why is data preprocessing important in machine learning?

Solution: Refer to Section 10.2

Q5 How is supervised learning similar to learning with a teacher?

Solution: Refer to Section 10.2

☛ Check Your Progress 2

Q1. What is Supervised Learning and how does it support predictive analytics?

Solution: Refer to Section 10.0

Q2. What are regression and classification problems in supervised learning?

Solution: Refer to Section 10.0

Q3. Differentiate between supervised and unsupervised learning..

Solution: Refer to Section 10.2

Q4. Explain the workflow of supervised learning.

Solution: Refer to Section 10.2

Q5. List and briefly explain any three supervised learning models

Solution: Refer to Section 10.3

Q6. What factors should be considered while selecting a supervised learning model?

Solution: Refer to Section 10.3

☛ Check Your Progress 3

Q1. What is the K-Nearest Neighbors (KNN) algorithm? Explain its basic principle..

Solution: Refer to Sub-section 10.3.1

Q2. Explain the working steps of the KNN algorithm.

Solution: Refer to Sub-section 10.3.1

Q3. What is the role of the parameter K in KNN?.

Solution: Refer to Sub-section 10.3.1

Q4. Why is feature scaling important in KNN?.

Solution: Refer to Sub-section 10.3.1

Q5. What are the advantages and limitations of KNN?

Solution: Refer to Sub-section 10.3.1

☛ Check Your Progress 4

Q1. What is the Naïve Bayes algorithm? Why is it called “naïve”?

Solution: Refer to Sub-section 10.3.2

Q2. State Bayes’ Theorem and explain its components.

Solution: Refer to Sub-section 10.3.2

Q3. Explain the working of the Naïve Bayes classifier.

Solution: Refer to Sub-section 10.3.2

Q4. What are the advantages of the Naïve Bayes algorithm?

Solution: Refer to Sub-section 10.3.2

Q5. What are the limitations of the Naïve Bayes algorithm?

Solution: Refer to Sub-section 10.3.2

☛ Check Your Progress 5

Q1. What is Logistic Regression? How is it different from Linear Regression?

Solution: Refer to Sub-section 10.3.3

Q2. Explain the sigmoid (logistic) function used in Logistic Regression.

Solution: Refer to Sub-section 10.3.3

Q3. How does Logistic Regression perform classification?

Solution: Refer to Sub-section 10.3.3

Q4. What are the advantages of Logistic Regression?.

Solution: Refer to Sub-section 10.3.3

Q5. What are the limitations of Logistic Regression?

Solution: Refer to Sub-section 10.3.3

Check Your Progress 6

Q1. What is a Decision Tree? Explain its basic structure.

Solution: Refer to Sub-section 10.3.4

Q2. How does a Decision Tree work in classification problems?

Solution: Refer to Sub-section 10.3.4

Q3. What are the advantages of Decision Trees?

Solution: Refer to Sub-section 10.3.4

Q4. What are the limitations of Decision Trees?.

Solution: Refer to Sub-section 10.3.4

Q5. What is overfitting in Decision Trees and how can it be controlled?

Solution: Refer to Sub-section 10.3.4

Check Your Progress 7

Q1. What is Random Forest? Explain its basic concept..

Solution: Refer to Sub-section 10.3.5

Q2. How does Random Forest improve over Decision Trees?

Solution: Refer to Sub-section 10.3.5

Q3. Explain the working of Random Forest..

Solution: Refer to Sub-section 10.3.5

Q4. What are the advantages of Random Forest?.

Solution: Refer to Sub-section 10.3.5

Q5. What are the limitations of Random Forest?

Solution: Refer to Sub-section 10.3.5

Check Your Progress 8

Q1. What is a Support Vector Machine (SVM)? Explain its objective..

Solution: Refer to Sub-section 10.3.6

Q2. What is a hyperplane and margin in SVM?

Solution: Refer to Sub-section 10.3.6

Q3. What are Support Vectors? Why are they important?

Solution: Refer to Sub-section 10.3.6

Q4. How does SVM handle non-linear data?

Solution: Refer to Sub-section 10.3.6

Q5. What are the advantages and limitations of SVM?

Solution: Refer to Sub-section 10.3.6

10.6 REFERENCES/FURTHER READINGS

1. *Pattern Recognition and Machine Learning* by Christopher M. Bishop, published by Springer (1st Edition, 2006, ISBN: 978-0387310732).
2. *Machine Learning* by Tom M. Mitchell, published by McGraw-Hill (1st Edition, 1997, ISBN: 978-0070428072).

3. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow* by Aurélien Géron, published by O'Reilly Media (2nd Edition, 2019, ISBN: 978-1492032649).
4. *Data Mining: Concepts and Techniques* by Jiawei Han, Micheline Kamber, and Jian Pei, published by Morgan Kaufmann (3rd Edition, 2011, ISBN: 978-0123814791)
5. *Practical Statistics for Data Scientists* by Peter Bruce, Andrew Bruce, and Peter Gedeck, published by O'Reilly Media (2nd Edition, 2020, ISBN: 978-1492072942).
6. *Business Analytics: Data Analysis and Decision Making* by S. Christian Albright and Wayne L. Winston, published by Cengage Learning (7th Edition, 2020, ISBN: 978-0357131787).
7. *Decision Analysis for Management Judgment* by Paul Goodwin and George Wright, published by Wiley (5th Edition, 2014, ISBN: 978-1119974017).
8. *Handbook of Statistical Analysis and Data Mining Applications* by Robert Nisbet, John Elder, and Gary Miner, published by Elsevier (2nd Edition, 2017, ISBN: 978-0124166455).
9. *Data Mining and Analysis: Fundamental Concepts and Algorithms* by Mohammed J. Zaki and Wagner Meira Jr., published by Cambridge University Press (1st Edition, 2014, ISBN: 978-0521766333).
10. *Core Concepts in Data Analysis: Summarization, Correlation and Visualization* by Boris Mirkin, published by Springer (1st Edition, 2011, ISBN: 978-0857292872).
11. *Applied Predictive Modeling*, Max Kuhn and Kjell Johnson, 1st Edition, Springer, 2013.
12. *An Introduction to Statistical Learning with Applications in R*, Gareth James, Daniela Witten, Trevor Hastie and Robert Tibshirani, 1st Edition, Springer, 2013.
13. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Trevor Hastie, Robert Tibshirani and Jerome Friedman, 2nd Edition, Springer, 2009.
14. *An Introduction to Statistical Learning with Applications in R* by Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani, published by Springer (2nd Edition, 2021, ISBN: 978-1071614174).
15. *Time Series Analysis: Forecasting and Control* by George E. P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung, published by Wiley (5th Edition, 2015, ISBN: 978-1118675021)