Supervised Learning (Regression/Classification): Basic Methods: Distance based Methods, Nearest Neighbours, Decision Trees, Naive Bayes, Linear Models: Linear Regression, Logistic Regression, Generalized Linear Models, Support Vector Machines, Binary Classification: Multiclass/Structured outputs, MNIST.

Supervised Learning(Regression/Classification)

Basic Methods: Distance based Methods, Nearest Neighbours, Decision Trees, Naive Bayes,

Linear Models: Linear Regression, Logistic Regression, Generalized Linear Models, Support Vector Machines

Binary Classification: Multiclass/Structured outputs, MNIST, Ranking.

1. Distance based Methods

- Distanced based algorithms are machine learning algorithms that classify queries by computing distances between these queries and a number of internally stored exemplars.

- Exemplars that are closets to the query have the largest influence on the classification assigned to the query.

In general there are 3 types of distance methods are there

1. Euclidean Distance
2. Manhattan Distance
3. Minkowski Distance

Euclidean Distance

- It is the most common use of distance. In most cases when people said about distance, they will refer to Euclidean distance. Euclidean distance is also known as simply distance.
- When data is dense or continuous, this is the best proximity measure. The Euclidean distance between two points is the length of the path connecting them. The Pythagorean theory gives distance between two points.
- Euclidean Distance $= \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

Manhattan Distance

- If you want to find Manhattan distance between two different points (x1, y1) and (x2, y2) such as the following, it would look like the following:
- Manhattan distance = $|x2 - x1| + |y2 - y1|$

Minkowski Distance

The generalized form of the Euclidean and Manhattan Distances is the Minkowski Distance. You can express the Minkowski distance as

$$D(X,Y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}$$

The order of the norm is represented by p. When an order(p) is 1, Manhattan Distance is represented, and when order(p) is 2 in the above formula, Euclidean Distance is represented.

Some frequently used distance functions.

Camberra :

$$d(x,y) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|} \qquad (2)$$

Minkowsky :

$$d(x,y) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r} \qquad (3)$$

Chebychev :

$$d(x,y) = \max_{i=1}^{m} |x_i - y_i| \qquad (4)$$

Euclidean :

$$d(x,y) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2} \qquad (5)$$

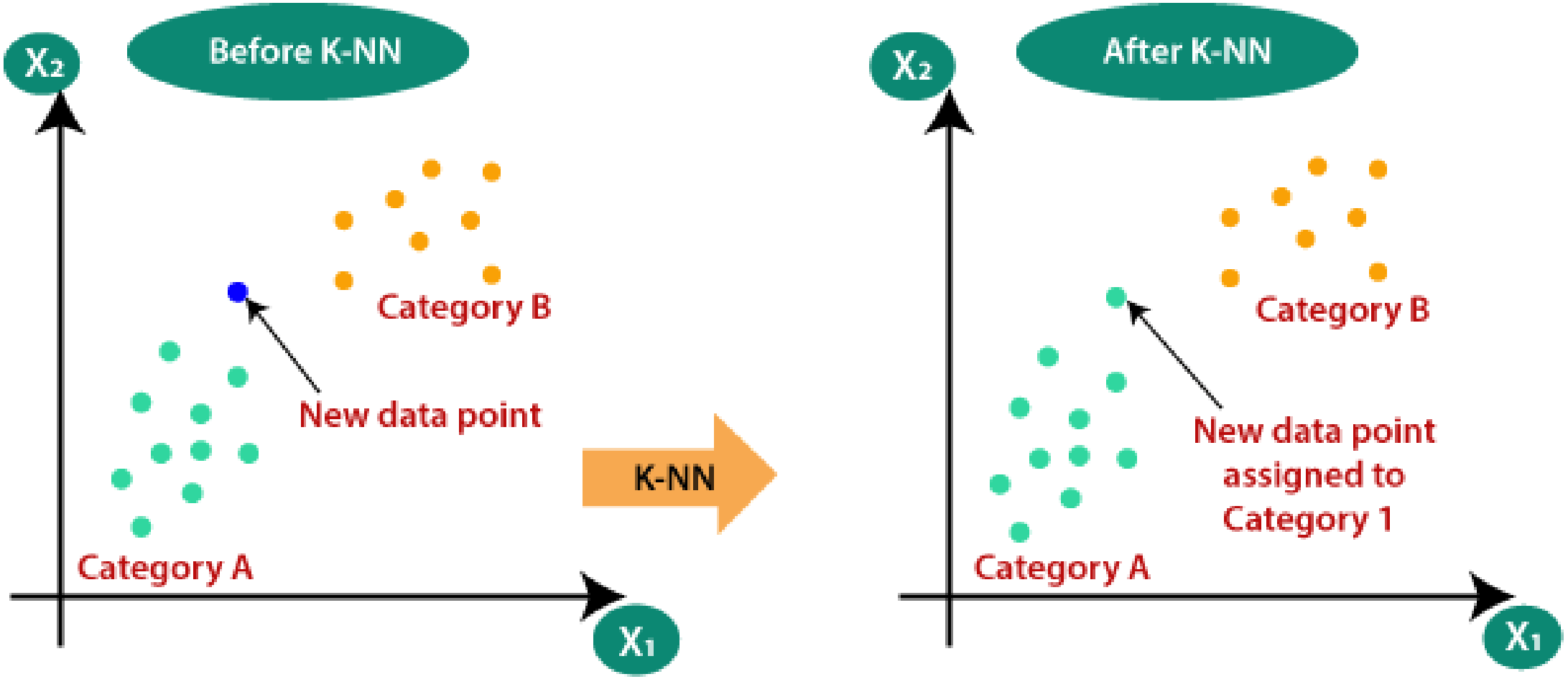
Manhattan / city - block :

$$d(x,y) = \sum_{i=1}^{m} |x_i - y_i| \qquad (6)$$

2. Nearest Neighbours

- The abbreviation KNN stands for “K-Nearest Neighbour”. It is a supervised machine learning algorithm.
- The algorithm can be used to solve both classification and regression problem statements.
- The number of nearest neighbours to a new unknown variable that has to be predicted or classified is denoted by the symbol ‘K’.

- K-NN is a non-parametric algorithm, which means it does not make any assumption on underlying data.
- It is also called a lazy learner algorithm because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.


Before K-NN
X2
Category B
New data point
Category A
X1
K-NN
After K-NN
X2
Category B
New data point
assigned to
Category 1
Category A
X1

How KNN Works:

1. **Choose a value for K** (the number of neighbors).
2. **Calculate the distance** between the query data point and all points in the dataset (commonly using Euclidean distance).
3. **Select the K nearest neighbors** to the query point.
4. **Assign a class** to the query point based on a majority vote of the K neighbors (for classification) or take the average of their values (for regression).

Example: Classifying a New Point with KNN

Let’s use an example dataset of fruits based on their weight and texture to classify whether a new fruit is an **apple** or an **orange**.

Weight (grams)	Texture (0 = Smooth, 1 = Rough)	Fruit (Label)
150	0	Apple
170	0	Apple
140	0	Apple
130	1	Orange
160	1	Orange
155	1	Orange

Now, we want to classify a fruit that weighs **145 grams** and has a **smooth texture (0)**.

Weight (grams)	Texture (0 = Smooth, 1 = Rough)	Fruit (Label)
150	0	Apple
170	0	Apple
140	0	Apple
130	1	Orange
160	1	Orange
155	1	Orange

Step-by-Step Process:

1. **Choose K = 3** (for this example).
2. Calculate the Euclidean distance between the new point (145, 0) and all existing points in the dataset:

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

For each point:

- Distance to (150, 0) = $\sqrt{((145 - 150)^2 + (0 - 0)^2)} = 5$
- Distance to (170, 0) = $\sqrt{((145 - 170)^2 + (0 - 0)^2)} = 25$
- Distance to (140, 0) = $\sqrt{((145 - 140)^2 + (0 - 0)^2)} = 5$
- Distance to (130, 1) = $\sqrt{((145 - 130)^2 + (0 - 1)^2)} = \sqrt{(15^2 + 1^2)} = \sqrt{226} \approx 15.03$
- Distance to (160, 1) = $\sqrt{((145 - 160)^2 + (0 - 1)^2)} = \sqrt{(15^2 + 1^2)} = \sqrt{226} \approx 15.03$
- Distance to (155, 1) = $\sqrt{((145 - 155)^2 + (0 - 1)^2)} = \sqrt{(10^2 + 1^2)} = \sqrt{101} \approx 10.05$

3. **Find the 3 nearest neighbors**: The closest points are:
    - (150, 0) with a distance of 5 (Apple)
    - (140, 0) with a distance of 5 (Apple)
    - (155, 1) with a distance of 10.05 (Orange)
4. **Classify the new point** based on a majority vote. Out of the 3 nearest neighbors, 2 are **Apples**, and 1 is **Orange**. Therefore, the new point is classified as an **Apple**.

Sepal Length	Sepal Width	Species
5.3	3.7	Setosa
5.1	3.8	Setosa
7.2	3.0	Virginica
5.4	3.4	Setosa
5.1	3.3	Setosa
5.4	3.9	Setosa
7.4	2.8	Virginica
6.1	2.8	Verscicolor
7.3	2.9	Virginica
6.0	2.7	Verscicolor
5.8	2.8	Virginica
6.3	2.3	Verscicolor
5.1	2.5	Verscicolor
6.3	2.5	Verscicolor
5.5	2.4	Verscicolor

Step 1: Find Distance

Sepal Length	Sepal Width	Species
5.2	3.1	?

Sepal Length	Sepal Width	Species	Distance	Rank
5.3	3.7	Setosa	0.608	3
5.1	3.8	Setosa	0.707	6
7.2	3.0	Virginica	2.002	13
5.4	3.4	Setosa	0.36	2
5.1	3.3	Setosa	0.22	1
5.4	3.9	Setosa	0.82	8
7.4	2.8	Virginica	2.22	15
6.1	2.8	Verscicolor	0.94	10
7.3	2.9	Virginica	2.1	14
6.0	2.7	Verscicolor	0.89	9
5.8	2.8	Virginica	0.67	5
6.3	2.3	Verscicolor	1.36	12

Step 3: Find the Nearest Neighbor

If k = 1 – Setosa

If k = 2 – Setosa

If k = 5 – Setosa

When to Consider Nearest Neighbors

- Instances map to points in R^N
- Less than 20 attributes per instance
- Lots of training data

Advantages:

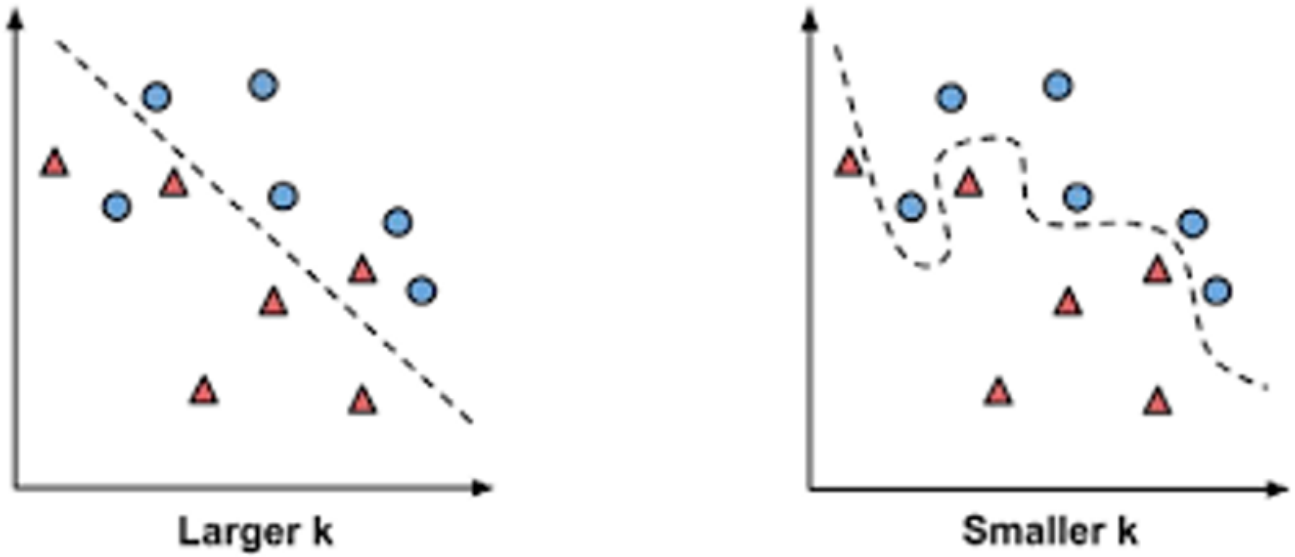
- Training is very fast
- Learn complex target functions
- Do not loose information

Disadvantages:

- Slow at query time (need good indexing)
- Irrelevant attributes may increase the distance of "truly" similar examples

The impact of selecting a smaller or larger K value on the model

- **Larger K value:** The case of underfitting occurs when the value of k is increased. In this case, the model would be unable to correctly learn on the training data.
- **Smaller k value:** The condition of overfitting occurs when the value of k is smaller. The model will capture all of the training data, including noise. The model will perform poorly for the test data in this scenario.



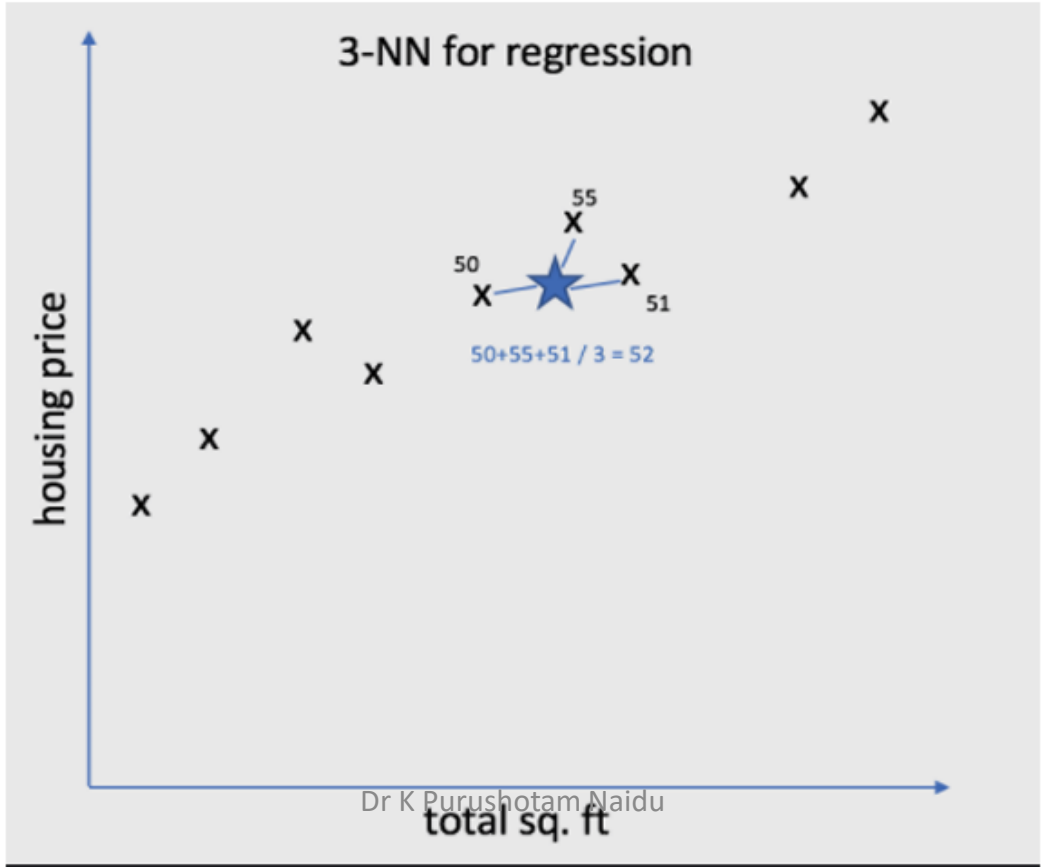
Src: https://images.app.goo.gl/vXStNS4NeEqUCDXn8

☞ Regression

KNN employs a mean/average method for predicting the value of new data. Based on the value of K, it would consider all of the nearest neighbours.

The algorithm attempts to calculate the mean for all the nearest neighbours' values until it has identified all the nearest neighbours within a certain range of the K value.

Consider the diagram below, where the value of k is set to 3. It will now calculate the mean (52) based on the values of these neighbours (50, 55, and 51) and allocate this value to the unknown data.

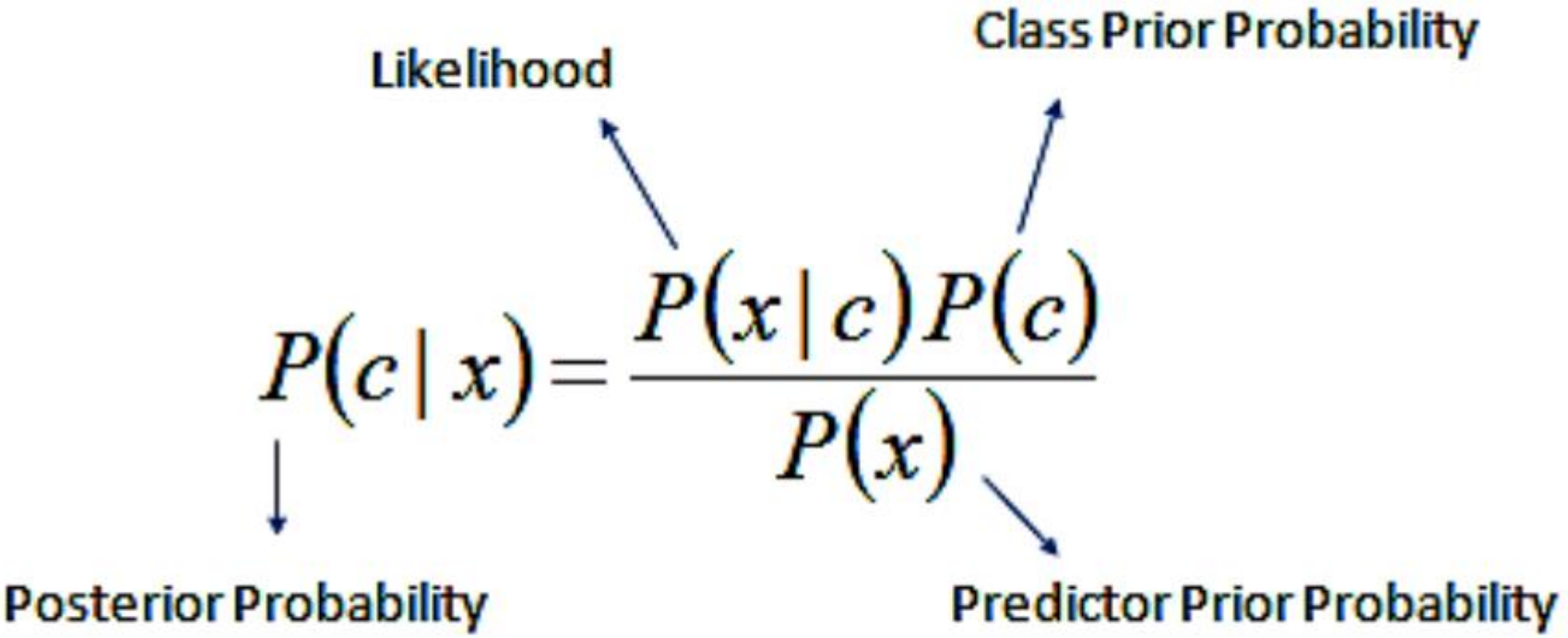


Naive Bayes algorithm

- **Naive Bayes algorithm** is a Supervised Machine Learning algorithm
- It is used to solve classification and regression problems
- The **Naive Bayes algorithm** is a probabilistic machine learning algorithm based on applying **Bayes' Theorem** with the "naive" assumption that the features are independent of each other.
- Despite its simplicity, Naive Bayes performs well in many practical applications, particularly for text classification (spam filtering, sentiment analysis) and categorical data prediction.

Bayes' Theorem

Bayes' Theorem is the foundation of Naive Bayes and is expressed as:



$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

- $P(c|x)$ is the **posterior probability**: the probability of class c given the feature x.
- $P(x|c)$ is the **likelihood**: the probability of observing x given that the class is c.
- $P(c)$ is the **prior probability**: the probability of class c occurring overall.
- $P(x)$ is the **evidence** or **marginal likelihood**: the total probability of observing x,

Naive Bayes Example: Weather Dataset

Consider a weather dataset used to predict whether to play **tennis** (Yes/No) based on these features: **Outlook**, **Temperature, Humidity and Windy**.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

We want to predict whether to play tennis on a day with:

Outlook = Sunny, **Temperature = Cool**, **Humidity = High, Windy=True**

- We will apply Naive Bayes to calculate the probability of **PlayTennis = Yes** and **PlayTennis = No** given this new data.
- The posterior probability can be calculated by first, constructing a frequency table for each attribute against the target. Then, transforming the frequency tables to likelihood tables and finally use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Step 1: Calculate Prior Probabilities

The prior probability is the fraction of days with **PlayTennis = Yes** or **PlayTennis = No**:

$$P(\text{Yes}) = \frac{9}{14}, \quad P(\text{No}) = \frac{5}{14}$$

Step 2: Calculate Likelihoods for Each Class

We calculate the likelihood of each feature given the class **Yes** or **No**.

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Frequency Table

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

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1

Likelihood Table

		Play Golf	
		Yes	No
Outlook	Sunny	3/9	2/5
	Overcast	4/9	0/5
	Rainy	2/9	3/5

		Play Golf	
		Yes	No
Humidity	High	3/9	4/5
	Normal	6/9	1/5

Outlook	Temp	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1

		Play Golf	
		Yes	No
Temp.	Hot	2/9	2/5
	Mild	4/9	2/5
	Cool	3/9	1/5

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3

		Play Golf	
		Yes	No
Windy	False	6/9	2/5
	True	3/9	3/5

Outlook	Temp	Humidity	Windy	Play
Rainy	Cool	High	True	?

Step 3: Calculate Posterior Probabilities

Using Bayes' Theorem, we compute the posterior probability for **Yes** and **No**:

Let X= (Outlook = Rainy, Temperature = Cool, Humidity = High, Windy=True)

Calculate P(Yes/X) as Follows:

$$P(Yes \mid X) = P(Rainy \mid Yes) \times P(Cool \mid Yes) \times P(High \mid Yes) \times P(True \mid Yes) \times P(Yes)$$

$$P(Yes \mid X) = 2/9 \times 3/9 \times 3/9 \times 3/9 \times 9/14 = 0.00529$$

Similarly Calculate P(No/X) as Follows:

$$P(No \mid X) = P(Rainy \mid No) \times P(Cool \mid No) \times P(High \mid No) \times P(True \mid No) \times P(No)$$

$$P(No \mid X) = 3/5 \times 1/5 \times 4/5 \times 3/5 \times 5/14 = 0.02057$$

Probability P(No/X) is greater than P(Yes/X) so the given test instance X belongs to NO class

Advantages of the Naive Bayes Algorithm:

1. **Simple and Easy to Implement**:
   - Naive Bayes is straightforward and easy to understand and implement. It requires fewer parameters to estimate and is ideal for small datasets.
2. **Fast and Efficient**:
   - Naive Bayes is computationally efficient and works well with large datasets. It requires minimal training time compared to other algorithms, such as decision trees or support vector machines.
3. **Works Well with High-Dimensional Data**:
   - It performs well when the input data has many features. This makes it ideal for applications like text classification, where there are many words or phrases (features) in the dataset.
4. **Handles Categorical Data Well**:
   - Naive Bayes is effective for categorical input variables. It can easily handle binary, nominal, or ordinal data.

5. **Requires Less Training Data**:
    - Since it makes strong independence assumptions, Naive Bayes can often perform well with a smaller training dataset, compared to more complex models.
6. **Performs Well with Multiple Classes**:
    - Naive Bayes is good at handling multi-class classification problems, where the task involves more than two possible output classes.
7. **Performs Well Even with Irrelevant Features**:
    - Naive Bayes works reasonably well even when some features are not informative, since it assumes that each feature contributes independently.
8. **Robust to Missing Data**:
    - The model can ignore missing values during probability calculations, allowing it to make predicti even with incomplete data.

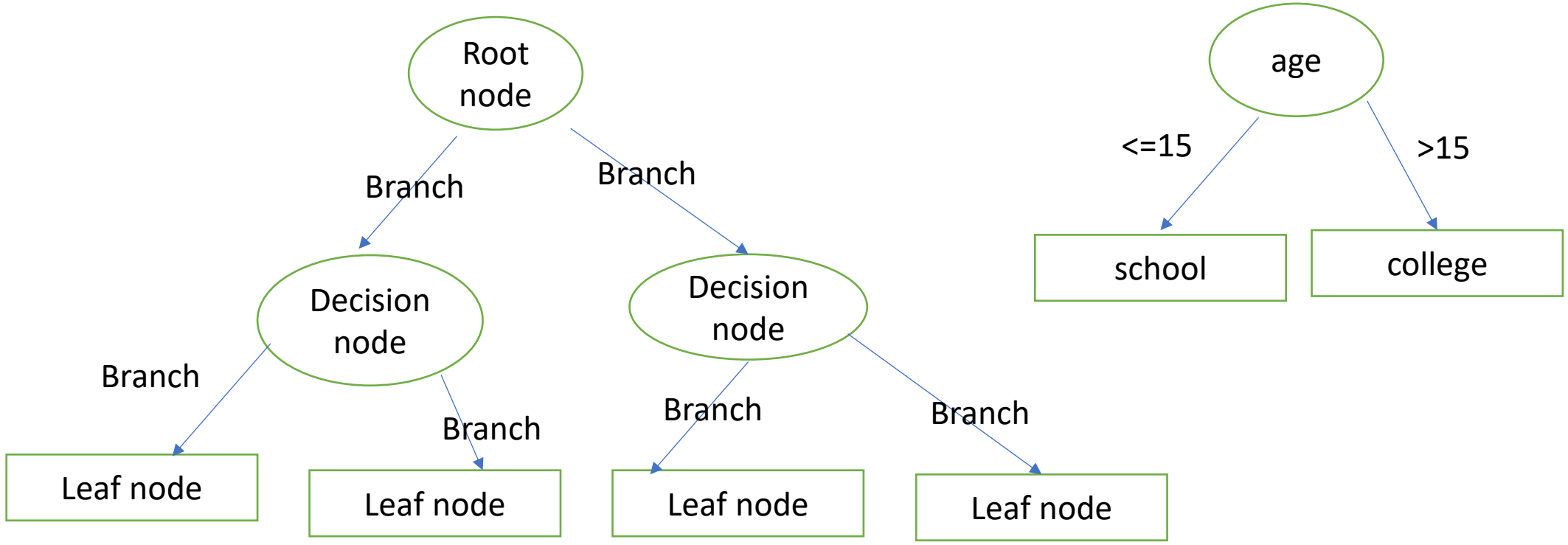
Disadvantages of the Naive Bayes Algorithm:

1. **Strong Independence Assumption**:
    - Naive Bayes assumes that all features are independent of each other, which is rarely true in real-world datasets. If features are highly correlated, the algorithm's performance can suffer.
2. **Zero Probability Problem**:
    - If the training dataset has never encountered a particular feature-value pair in a class, Naive Bayes assigns it a zero probability. This can be handled by techniques like **Laplace smoothing**, but it is still a limitation.
3. **Not Suitable for Continuous Data Without Discretization**:
    - Naive Bayes assumes that all features are categorical. It doesn't work well with continuous data unless it is discretized or handled through Gaussian Naive Bayes, which assumes the data follows a normal distribution.

4. **Limited Expressiveness**:
    - Since it makes simplistic assumptions about data distributions, Naive Bayes cannot capture complex relationships between features. Other algorithms, like decision trees or neural networks, may outperform Naive Bayes when relationships between features are intricate.
5. **Sensitivity to Skewed Data**:
    - If the dataset is unbalanced, meaning that certain classes occur far more frequently than others, Naive Bayes can be biased towards the majority class.
6. **Poor Performance on Highly Complex Problems**:
    - For tasks requiring understanding of interactions between variables, such as image recognition or high-dimensional structured data, Naive Bayes typically underperforms compared to more sophisticated algorithms like Random Forests, SVMs, or Neural Networks.

Decision Trees

- A **Decision Tree** is a supervised learning algorithm that can be used for both **classification** and **regression** tasks.



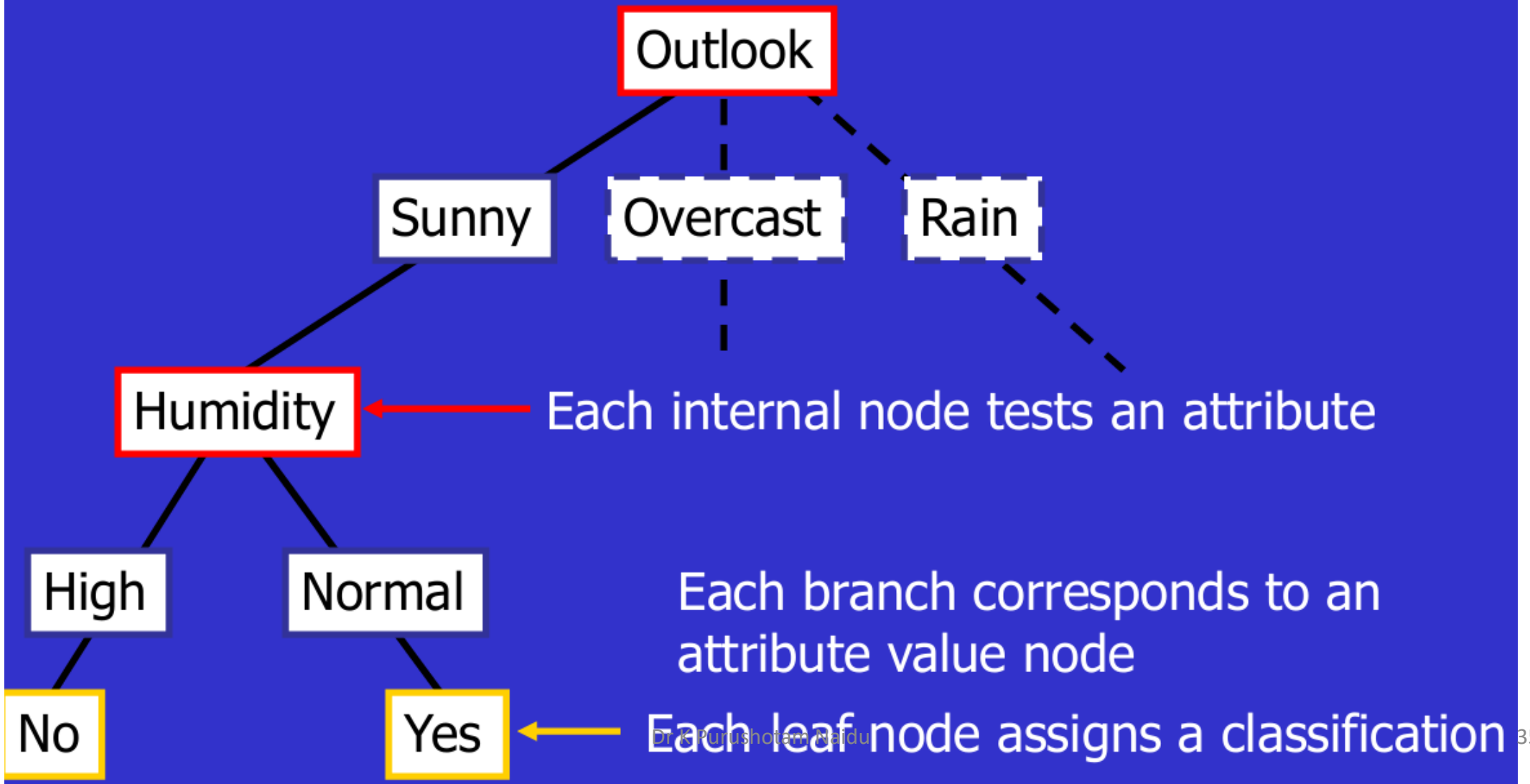
Key Concepts of Decision Trees

1. **Decision Nodes**: Nodes where the dataset is split based on a certain feature.
2. **Leaf Nodes**: Terminal nodes that represent the final classification or regression value.
3. **Branches**: These are connections between nodes, representing the outcome of a decision at a particular node.

Training Examples

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Decision Tree for PlayTennis



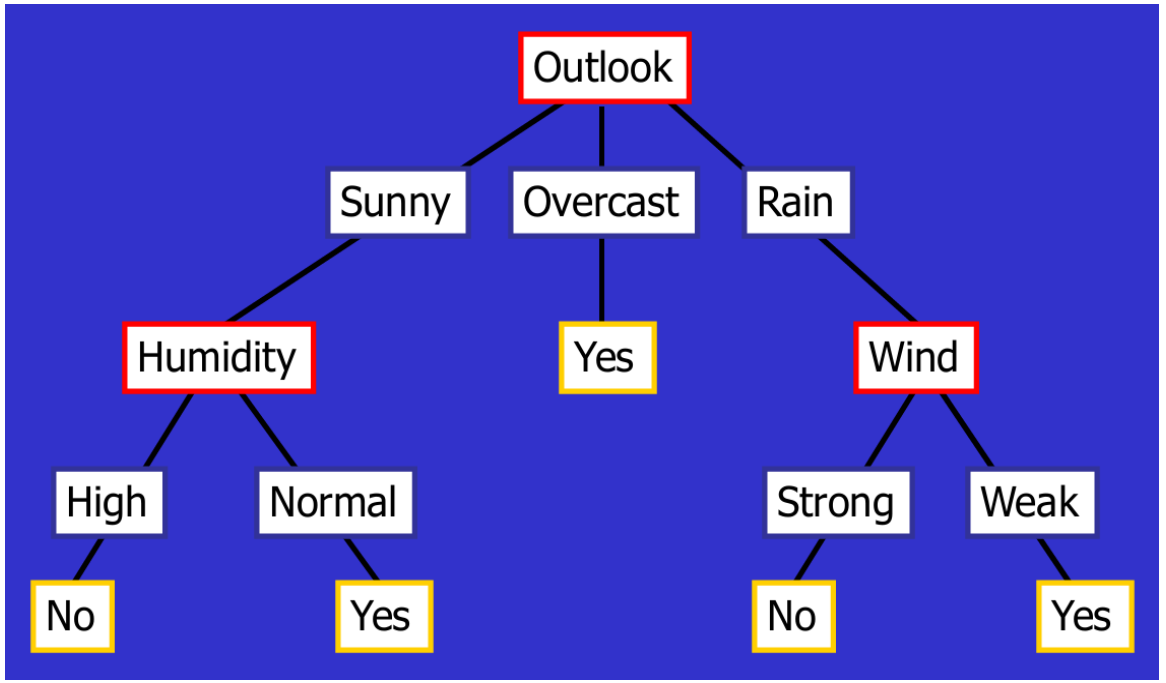
- Decision trees are built using algorithms such as
  - **ID3 (Iterative Dichotomiser 3)**
  - **CART (Classification and Regression Trees)**
  - **C4.5.**
- Entropy, Information Gain, Gini Index, and Gain Ratio are metrics helps to determine the best attribute to split on at each node in decision tree algorithms
- Entropy and Information Gain are popular with algorithms like ID3 and C4.5, while the Gini Index is used in CART. Gain Ratio, used in C4.5, is beneficial when working with attributes that have many levels.

Training Examples

Day	Outlook	Temp.	Humidity	Wind	Play Tennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Weak	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Strong	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Metrics commonly used to determine best attribute in decision tree

- Entropy
- Gini Index
- Information Gain
- Gain Ratio



These are the metrics commonly used to measure the effectiveness of splits in decision tree algorithms, which help determine the best attribute to split on at each node. Let's look at each metric with formulas and examples

1. Entropy

- **Definition**: Entropy is a measure of the disorder or uncertainty in a dataset. In decision trees, entropy is used to evaluate how well a split separates the classes in the data.
- **Formula**: For a dataset with n classes, entropy $E(S)$ is defined as:

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

where p_i is the proportion of class i instances in the dataset S.

- **Example**: Suppose a dataset has 10 instances, with 6 instances of class A and 4 instances of class B.
  - Probability of class A, $p_A = \frac{6}{10} = 0.6$
  - Probability of class B, $p_B = \frac{4}{10} = 0.4$

$$E(S) = -(0.6 \log_2(0.6) + 0.4 \log_2(0.4)) \approx 0.971$$

- **Formula**:

$$\text{Gini Index}(S) = 1 - \sum_{i=1}^{n} p_i^2$$

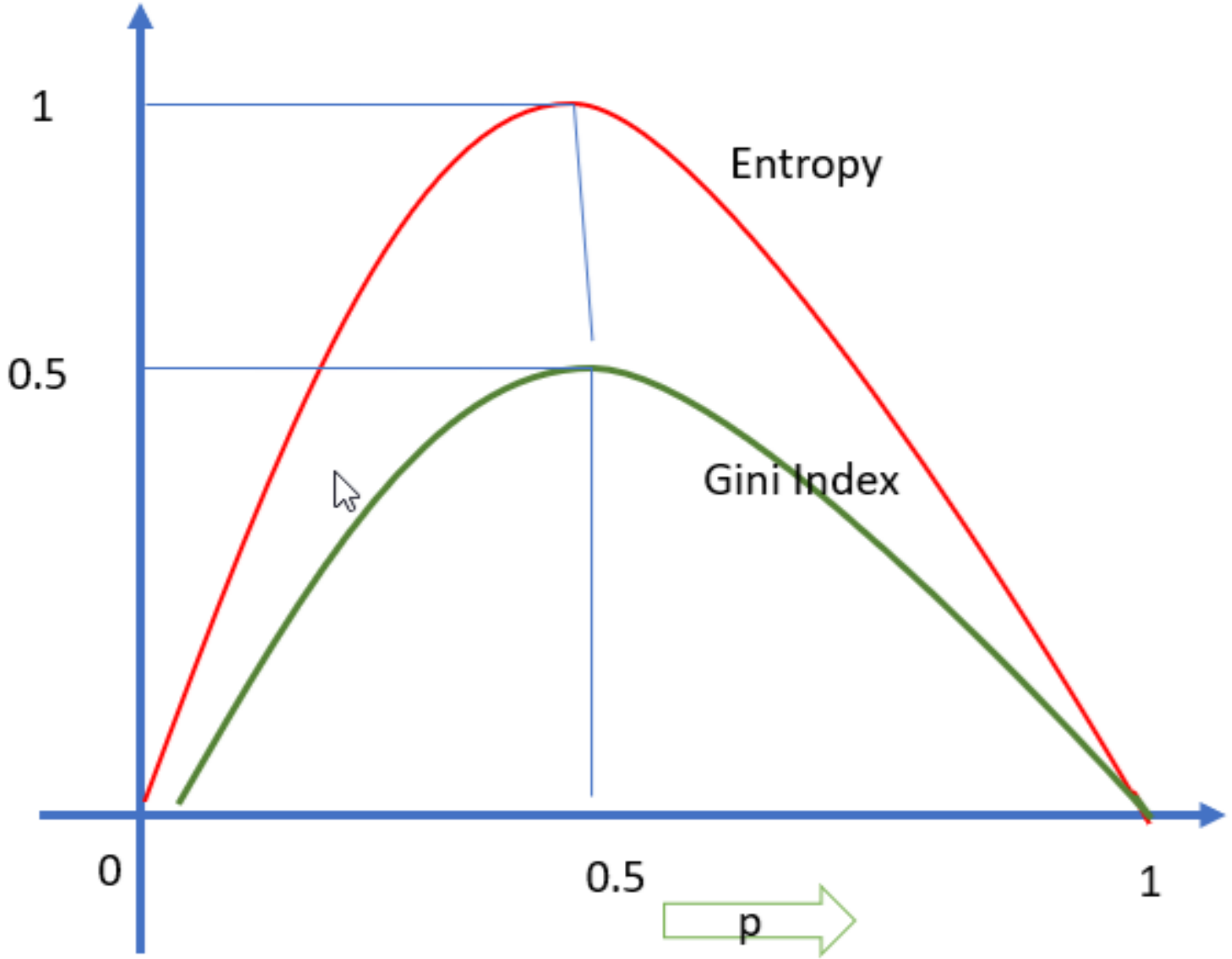
where p_i is the probability of class i in dataset S, and n is the number of classes.

- **Example**: Using the same dataset S with 6 positives and 4 negatives:
  - $p_{\text{positive}} = 0.6$
  - $p_{\text{negative}} = 0.4$

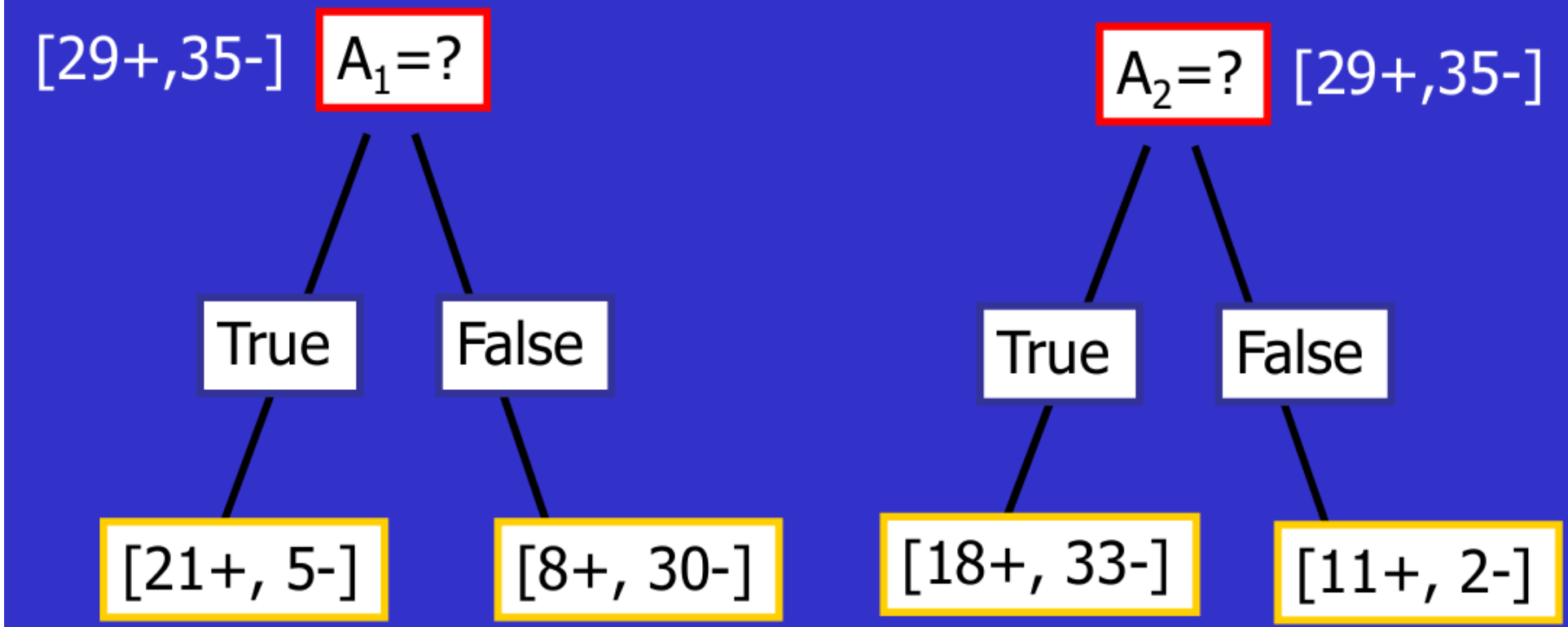
  Calculate the Gini Index:

$$\text{Gini Index}(S) = 1 - (0.6^2 + 0.4^2) = 1 - (0.36 + 0.16) = 1 - 0.52 = 0.48$$

- Entropy is 0 when the dataset is perfectly pure (i.e., all instances belong to a single class).
- Entropy is maximum when the dataset is evenly split among classes (e.g., 50% positive and 50% negative in a binary classification).
- Gini Index is 0 when all instances belong to one class (pure dataset).
- Gini Index is maximum (equal to 0.5 in binary classification) when classes are evenly distributed, i.e., the dataset is maximally impure.


1
0.5
Entropy
Gini Index
0
0.5
1
p

Which Attribute is ”best”?



. Information Gain

- **Definition**: Information Gain (IG) measures the reduction in entropy after a dataset is split on an attribute. It helps in selecting the attribute that maximizes information gain and best separates the classes.
- **Formula**:

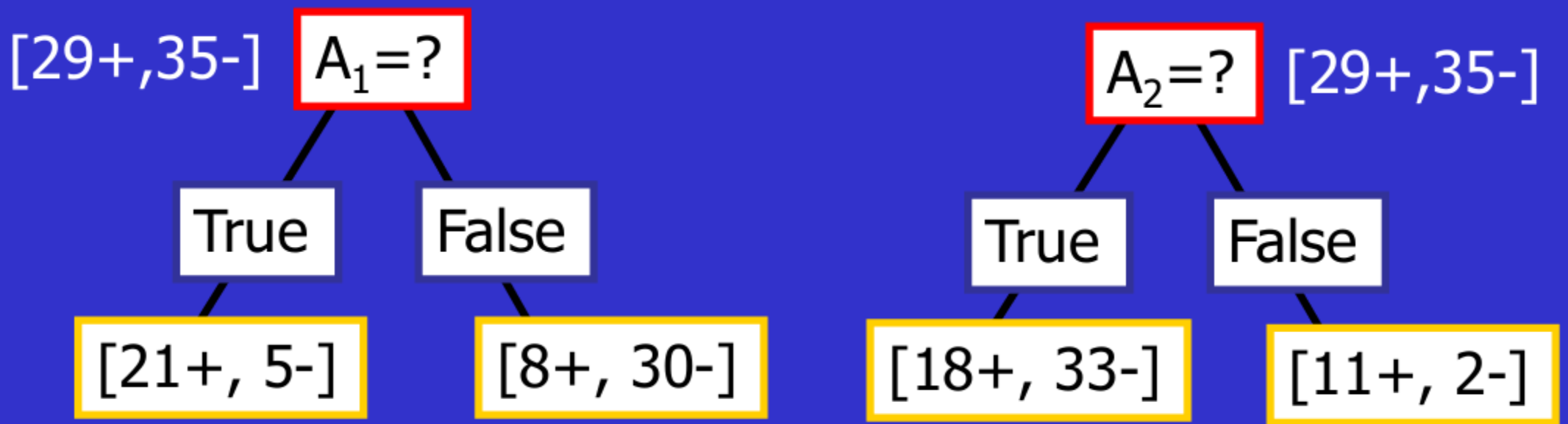
$$\text{Information Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

where S is the original dataset, A is the attribute, S_v is the subset of S where attribute A has value v, and $|S_v|$ is the number of instances in S_v.

- Gain(S,A): expected reduction in entropy due to sorting S on attribute A

$$Gain(S,A)=Entropy(S) - \sum_{v \in values(A)} |S_v|/|S| \; Entropy(S_v)$$

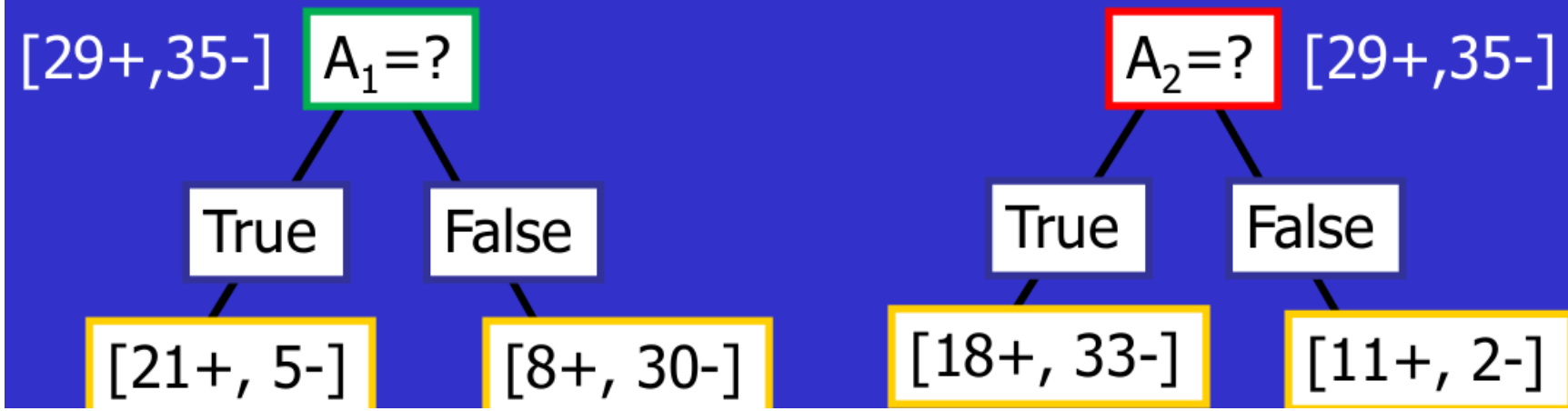
$$Entropy([29+,35-]) = -29/64 \log_2 29/64 - 35/64 \log_2 35/64 = 0.99$$



Information Gain

Entropy([21+,5-]) = 0.71
Entropy([8+,30-]) = 0.74
Gain(S,A_1)=Entropy(S)
-26/64*Entropy([21+,5-])
-38/64*Entropy([8+,30-])
=0.27

Entropy([18+,33-]) = 0.94
Entropy([11+,2-]) = 0.62
Gain(S,A_2)=Entropy(S)
-51/64*Entropy([18+,33-])
-13/64*Entropy([11+,2-])
=0.12



- **Formula**:

$$\text{Gain Ratio}(S, A) = \frac{\text{Information Gain}(S, A)}{\text{Split Information}(S, A)}$$

where

$$\text{Split Information}(S, A) = -\sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \log_2 \left(\frac{|S_v|}{|S|} \right)$$

Here, Split Information measures the potential information generated by splitting the dataset S on attribute A.

- **Example**: Continuing with the example:
  - Suppose the Information Gain of attribute A is 0.128.
  - Calculate Split Information for A:

$$\text{Split Information}(S, A) = -\left(\frac{5}{10} \log_2 \left(\frac{5}{10} \right) + \frac{5}{10} \log_2 \left(\frac{5}{10} \right) \right) = 1$$

Now, compute Gain Ratio:

$$\text{Gain Ratio}(S, A) = \frac{0.128}{1} = 0.128$$

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Weather Dataset

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Entropy of target variable (play tennis) of weather dataset ①

$$\text{Entropy}(S) = -P_{yes}\log_2(P_{yes}) - P_{NO}\log_2(P_{NO})$$

$$= -\frac{9}{14}\log_2(9/14) - \frac{5}{14}\log_2(5/14)$$

$$= 0.409 - (-.5305)$$

$$= 0.409 + .5305$$

$$= 0.9395$$

$$\text{Entropy}(S) \approx 0.94$$

$$\text{Entropy}(S) = -\sum_{i=1}^{K} P_i \log_2 P_i$$

where K is no. of classes

P_i is the probability of class i in the dataset S.

Information Gain for Outlook

$$IG = \text{Entropy}_{Before} - \sum_{i=1}^{K} \frac{|S_i|}{|S|}\text{Entropy}(S_i)$$

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Outlook = Sunny $\Rightarrow$

$$\text{Entropy (sunny)} = \frac{-2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}$$

$$= 0.971$$

Outlook = Overcast $\Rightarrow$

$$\text{Entropy (overcast)} = \frac{-4}{4}\log_2 4/4 - 0 = 0$$

Outlook = Rainy $\Rightarrow$

$$\text{Entropy (Rainy)} = \frac{-3}{5}\log_2 3/5 - 2/5\log_2 2/5$$

$$= 0.971$$

$$IG(\text{outlook}) = 0.94 - \left[\frac{5}{14}\times 0.971 + \frac{4}{14}\times 0 + \frac{5}{14}\times 0.971\right]$$

$$= 0.246$$

$$\approx 0.25$$

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Information Gain for temp

$$IG(temp) = 0.94 - \left[\frac{4}{14}(1) + \frac{4}{14}\left[\frac{-3}{4}\log_2 3/4 - \frac{1}{4}\log_2 1/4\right] + \frac{6}{14}\left[\frac{-4}{6}\log_2 4/6 - \frac{2}{6}\log_2 2/6\right]\right]$$

$$= 0.0289$$

$$\approx 0.029$$

Information Gain for Humidity

$$IG(Hum) = 0.94 - \left[\frac{7}{14} \times \left[\frac{-3}{7}\log_2 3/7 - \frac{4}{7}\log_2 4/7\right] + \frac{7}{14} \times \left[\frac{-6}{7}\log_2 6/7 - 1/7\log_2 1/7\right]\right]$$

$$= 0.15154$$

$$\approx 0.1516$$

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Information Gain for Wind

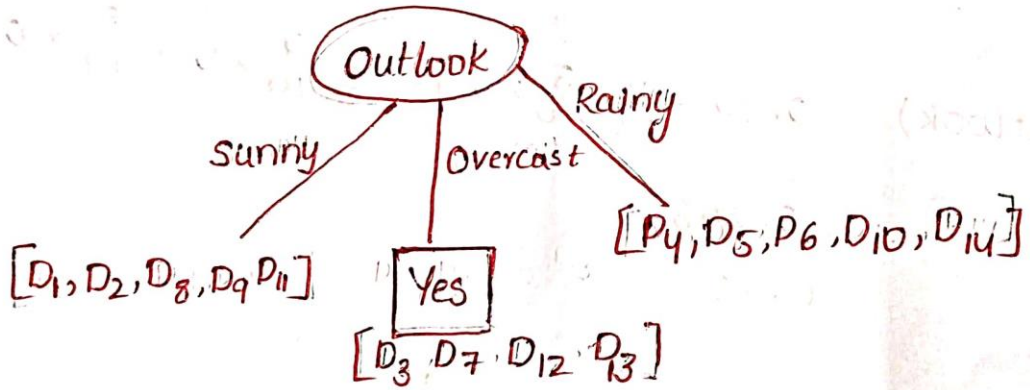
$$IG(wind) = 0.94 - \left[\frac{6}{14} \times \left[\frac{-3}{6}\log_2 3/6 - \frac{3}{6}\log_2 \frac{3}{6}\right] + \frac{8}{14} \times \left[\frac{-6}{8}\log_2 6/8 - \frac{2}{8}\log_2 2/8\right]\right]$$

$$= 0.94 - 0.892$$

$$= 0.048$$

Outlook has the highest Information Gain (i.e 0.25)

∴ root node is outlook



Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Here consider D1,D2,D8,D9,D11

$$E(\text{outlook} = \text{Sunny}) = -\frac{2}{5}\log_2 2/5 - \frac{3}{5}\log_2 3/5$$

$$= 0.971$$

$$IG(\text{Ol} = \text{sunny}, \text{temp}) = 0.971 - \left[\frac{2}{5} \times \left[\frac{-2}{2}\log_2 2/2 - 0\right] + \frac{2}{5} \times \left[\frac{-1}{2}\log_2 1/2 - \frac{1}{2}\log_2 1/2\right] + \frac{1}{5} \times \log 1\right]$$

$$= 0.971 - 0.4$$

$$= 0.571$$

$$IG(\text{Ol} = \text{sunny}, \text{Hum}) = 0.971 - \left[\frac{3}{5} \times \left[\frac{-0}{3}\log_2 0/3 - \frac{3}{3}\log_2 3/3\right] + \frac{2}{5} \times \left[\frac{-2}{2}\log_2 2/2 - 0\right]\right]$$

$$= 0.971$$

Decision Tree Construction using ID3 Algorithm

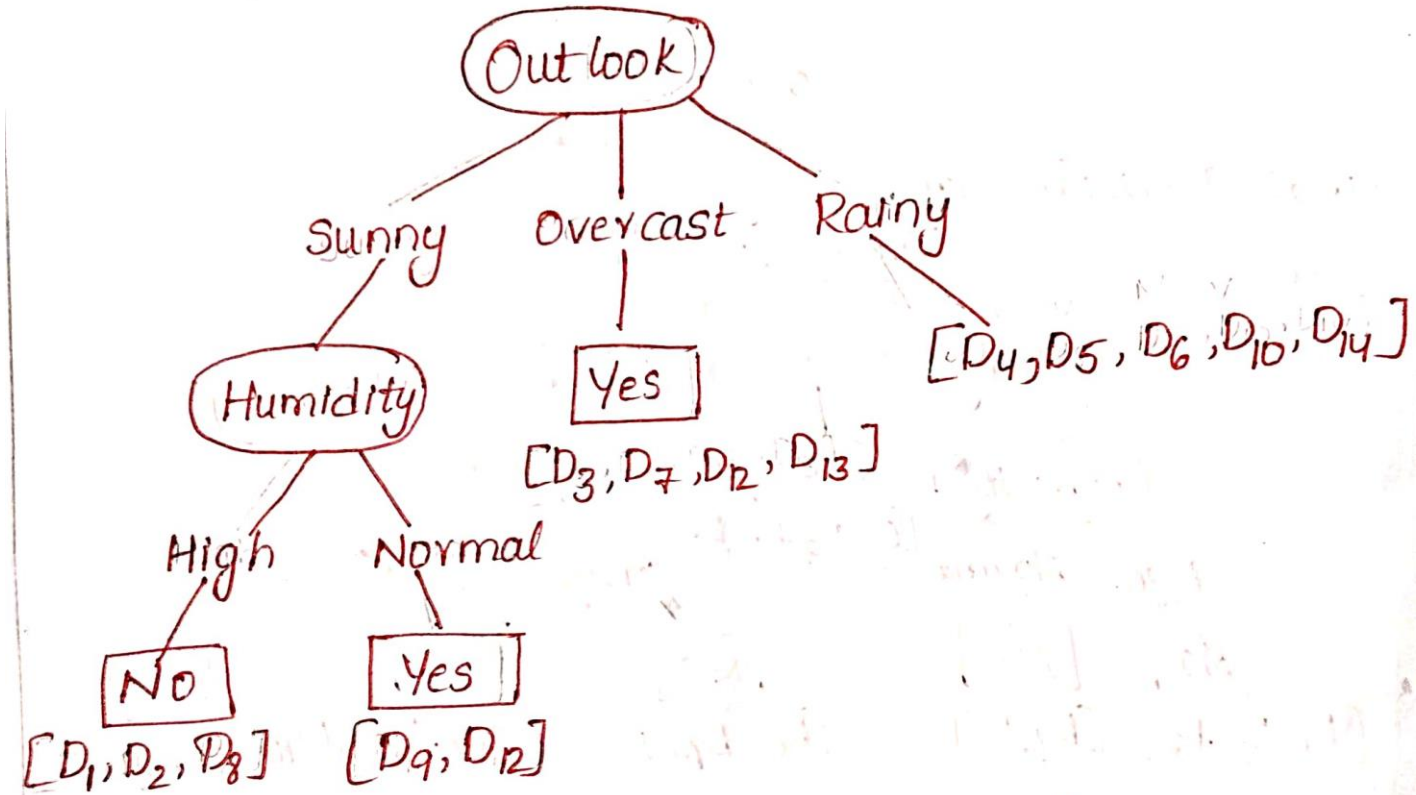
Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Here consider D1,D2,D8,D9,D11

$$IG(Ol=sunny, wind) = 0.971 - \left[\frac{2}{5} \times 1 + \frac{3}{5} \times \left[-\frac{1}{3}\log_2 1/3 - \frac{2}{3}\log_2 2/3\right]\right]$$

$$= 0.971 - 0.951$$

$$= 0.02$$



Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

Here consider D4,D5,D6,D10,D14

$E(\text{outlook} = \text{rainy}) = \frac{-3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2 2/5$ (A)

$= 0.971$

$IG(\text{ol} = \text{rainy}, \text{temp}) = 0.971 - \left[0 + \frac{3}{5} \times \left[\frac{-2}{3}\log_2 2/3 - \frac{1}{3}\log_2 1/3\right] + \frac{2}{5} \times \left[\frac{-1}{2}\log_2 1/2 - \frac{1}{2}\log_2 1/2\right]\right]$

$= 0.971 - 0.951$

$= 0.02$

$IG(\text{ol} = \text{rainy}, \text{hum}) = 0.971 - \left[\left[\frac{2}{5} \times \left[\frac{-1}{2}\log_2 1/2 - \frac{1}{2}\log_2 1/2\right] + \frac{3}{5} \times \left[\frac{-2}{3}\log_2 2/3 - 1/3\log_2 1/3\right]\right]\right.$

$= 0.971 - 0.951$

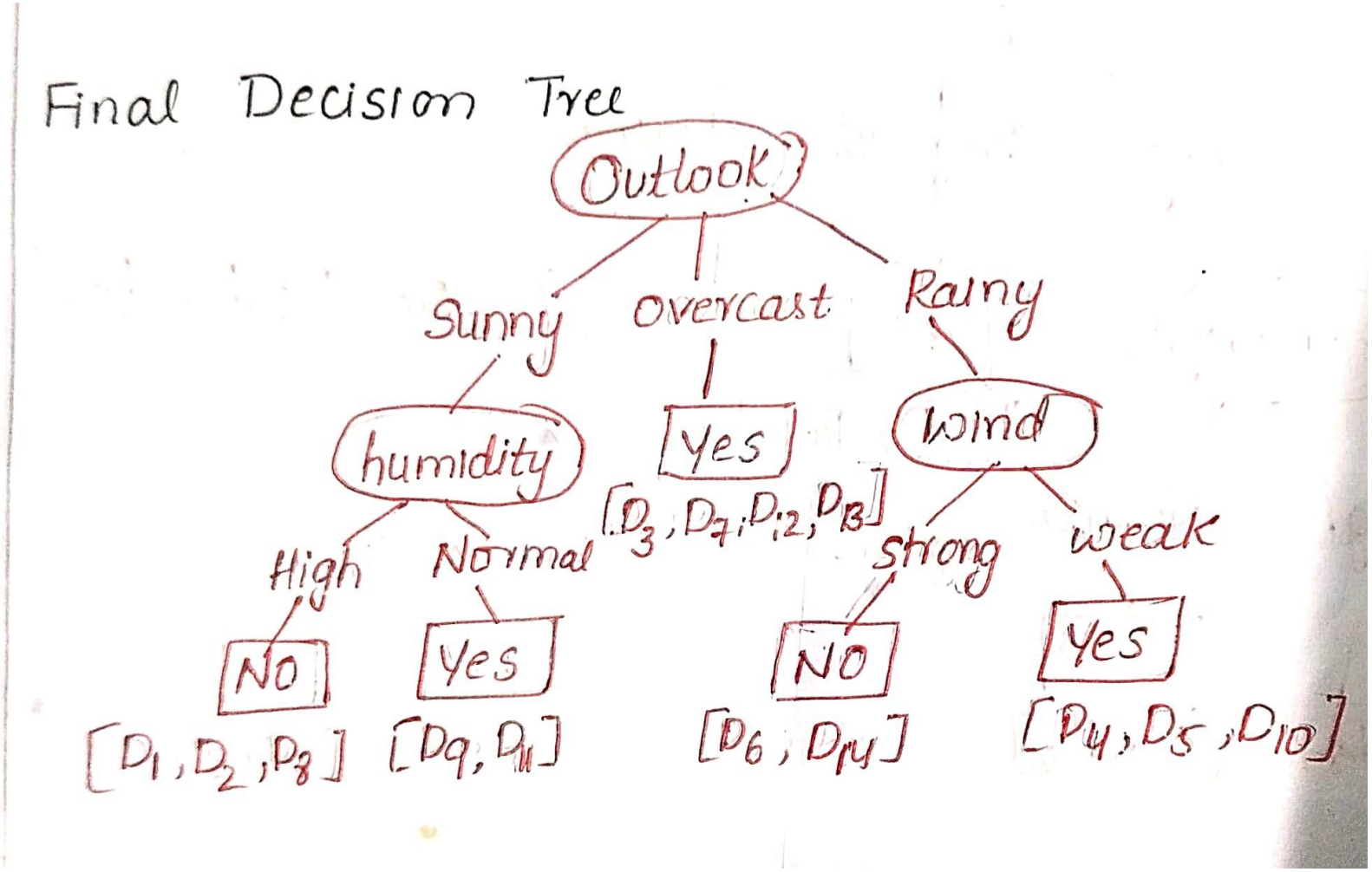
$= 0.02$

$IG(\text{ol} = \text{rainy}, \text{wind}) = 0.971 - \frac{2}{5} \times \left[0 \text{ or } -\frac{2}{2}\log_2 2/2\right] + \frac{3}{5} \times \left[\frac{-3}{3}\log_2 3/3 - 0\right]$

$= 0.971$

Decision Tree Construction using ID3 Algorithm

Day	Outlook	Temperature	Humidity	Wind	Play ball
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

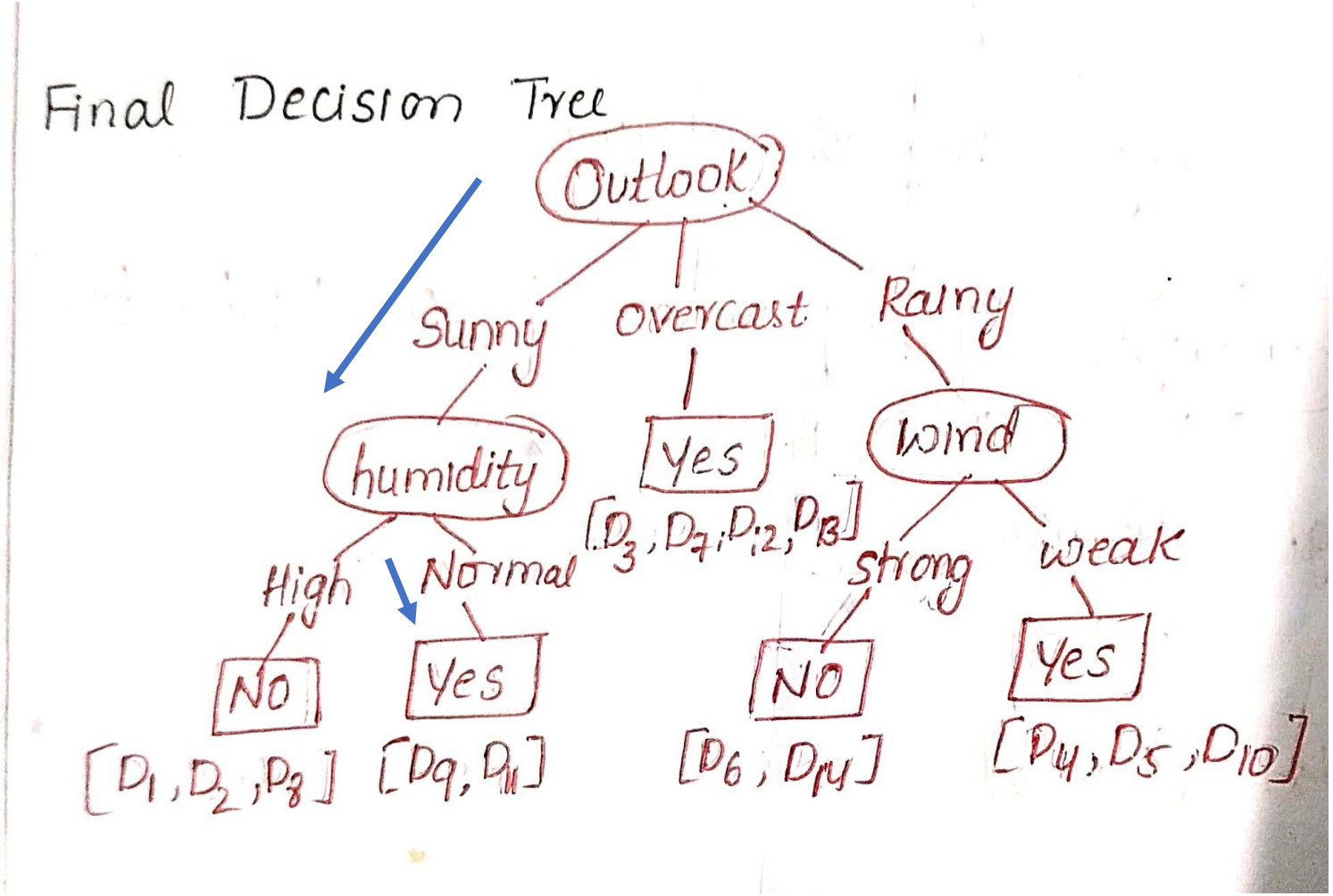


Predicting the class from the Decision Tress

For example predict the class of the following test instance:

Outlook=sunny, temp=hot
Humidity=Normal,
wind=weak

From the decision tree the class of given instance is "Yes"

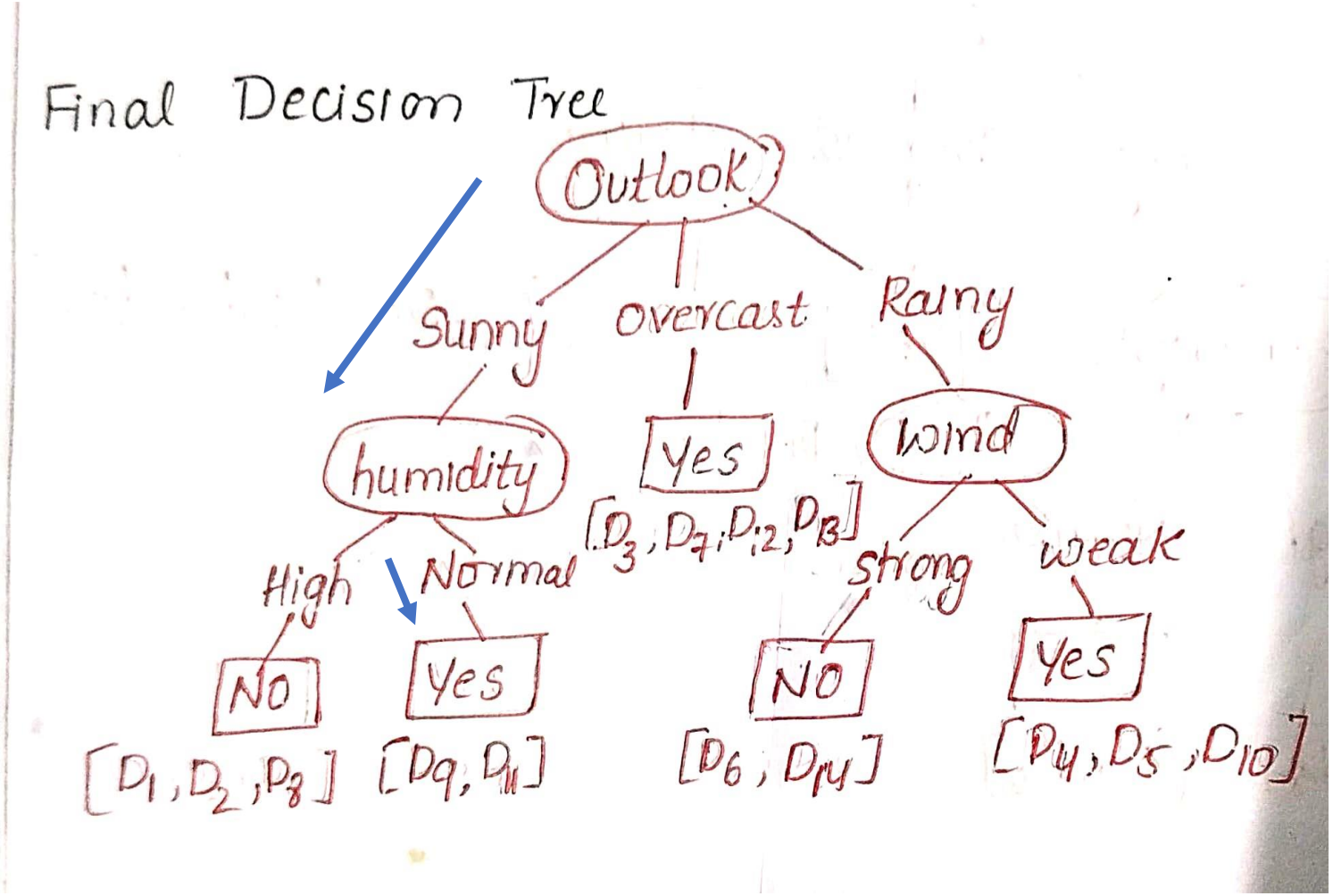


Predicting the class from the Decision Tress

For example predict the class of the following test instance:

Outlook=sunny, temp=hot
Humidity=Normal,
wind=weak

From the decision tree the class of given instance is "Yes"



ID3 algorithm for building a decision tree

1. Calculate Entropy for the Dataset

- Formula for entropy:

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

where p_i is the proportion of class i in the dataset S.

- Entropy = 0 when the data is perfectly pure (i.e., all instances belong to a single class).
- Entropy = 1 when the data is equally distributed between different classes (maximum impurity).

ID3 algorithm for building a decision tree

2. **Calculate Information Gain for Each Feature**:
   - For each feature, split the dataset based on its values.
   - Calculate the entropy for each subset and find the weighted average entropy after the split.
   - Information Gain (IG) for a feature A is:

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

   where S_v is the subset of S where feature A has value v.

3. **Select the Feature with the Highest Information Gain**:
   - Choose the feature with the highest IG as the root or decision node of the tree.

ID3 algorithm for building a decision tree

4. **Split the Dataset**:
   - Divide the dataset based on the values of the chosen feature.
5. **Repeat the Process Recursively for Each Subset**:
   - For each subset, remove the chosen feature and repeat steps 1–4, treating each subset as a new dataset.
   - Stop when all samples in a subset have the same label (pure subset) or when there are no remaining features.
6. **Create Leaf Nodes**:
   - If a subset is pure or you have no more features, assign a leaf node with the majority class label of that subset.

Repeat this process until the decision tree is fully grown. The result is a decision tree that can classify new instances based on the features.

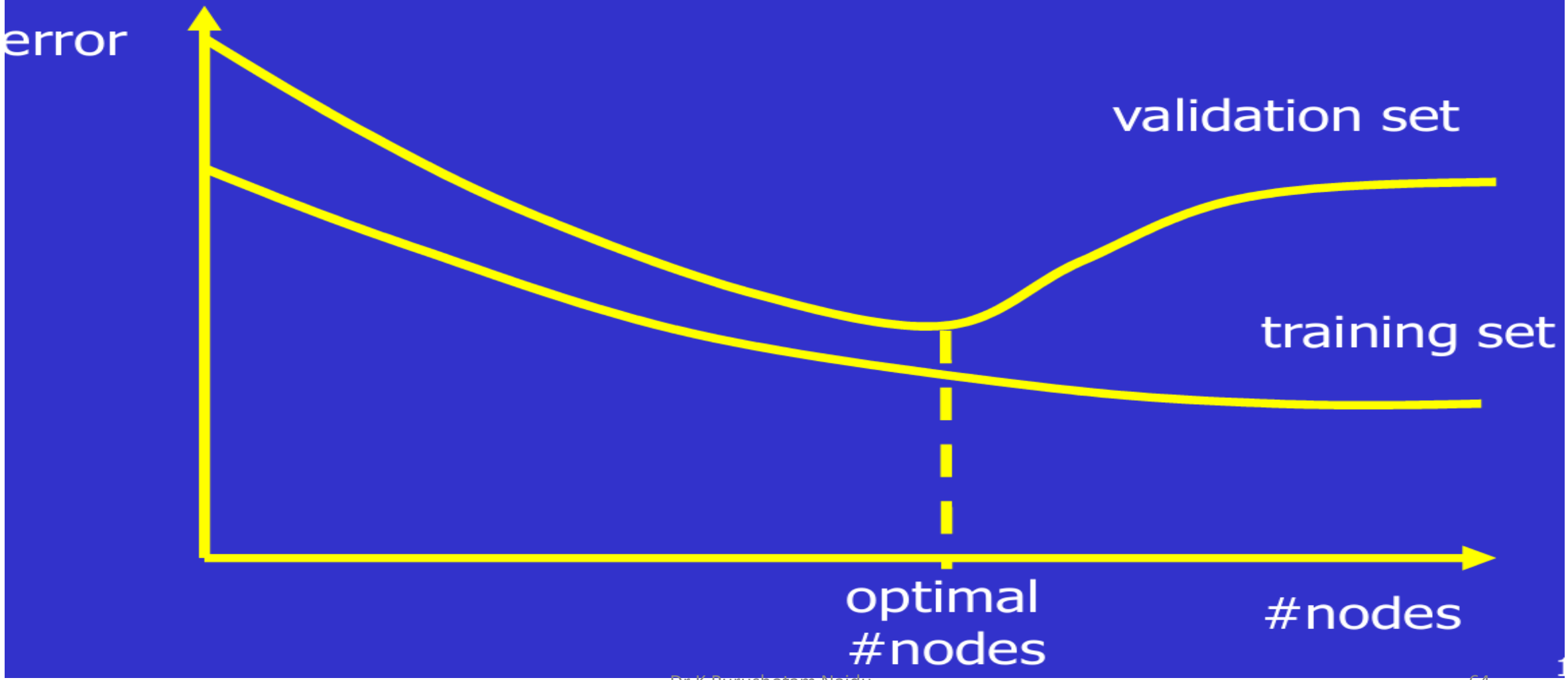
Advantages of Decision Trees:

1. **Interpretability**: Decision trees are easy to visualize and interpret, making them great for understanding and explaining models.
2. **No Need for Normalization**: They don't require scaling or normalization of features.
3. **Handles Categorical and Continuous Data**: Decision trees can handle both types of data without much preprocessing.
4. **Non-parametric**: They don't assume any underlying distribution of the data.

Disadvantages of Decision Trees:

1. **Overfitting**: Decision trees tend to overfit, especially if they are deep and complex. Pruning techniques can help combat this.
2. **Unstable**: Small changes in data can result in a completely different tree.
3. **Biased toward dominant classes**: They can become biased if some classes are more frequent than others.

Overfitting in Decision Trees



Techniques to avoid overfitting in decision trees

1. Pruning the Tree
   - **Pre-Pruning (Early Stopping)**:
     - Stop the growth of the tree before it becomes too complex
   - **Post-Pruning**:
     - Allow the tree to grow fully, then remove nodes that don't improve model performance significantly.

2. Set Maximum Features

- Limit the number of features considered at each split. For example, in the CART algorithm, you can limit the maximum number of features (`max_features`) used for splitting, which adds randomness and reduces the chance of overfitting.

3. Limit the Maximum Leaf Nodes

- Restrict the number of leaf nodes the tree can have by setting `max_leaf_nodes`. This prevents the tree from growing too many small branches and fitting noise.

Techniques to avoid overfitting in decision trees

4. Use Cross-Validation

- Use k-fold cross-validation to test the tree's performance on multiple subsets

5. Use Ensemble Methods (Bagging and Boosting)

- **Random Forest**: Combines multiple decision trees, each trained on random subsets of the data and features, to reduce overfitting. The aggregation of many models smooths out the individual trees' tendency to overfit.
- **Gradient Boosting**: Builds trees sequentially, correcting the errors of previous trees, and can use techniques like regularization to prevent overfitting.

6. Add Regularization

Regularization (Lasso), L2 Regularization (Ridge), Elastic Net

Techniques to avoid overfitting in decision trees

8. Handle Noisy Data

- Remove or reduce noisy data and outliers before training. Noisy data can cause the tree to overfit by creating branches that focus on anomalies rather than general patterns.

9. Use Feature Engineering

- Reduce the number of irrelevant or redundant features through proper feature selection. Fewer irrelevant features can make the tree simpler and less likely to overfit.

Parameters to Use in Decision Tree Libraries (e.g., scikit-learn):

- **max_depth**: Limit the depth of the tree.
- **min_samples_split**: Minimum number of samples to split a node.
- **min_samples_leaf**: Minimum number of samples in a leaf node.
- **max_features**: Number of features to consider when looking for the best split.
- **max_leaf_nodes**: Maximum number of leaf nodes allowed in the tree.

Strengths & Weaknesses of DT

Strengths include:

- ☑ Fast and simple to implement
- ☑ Can outperform human experts in many problems
- ☑ Easy to interpret (=rules)
- ☑ Empirically valid in many commercial products
- ☑ Handles noisy data - robust to outliers
- ☑ Automatic feature selection

Weaknesses include:

- ☒ "Univariate" splits/partitioning using only one attribute at a time so limits types of possible trees
- ☒ Large decision trees may be hard to understand
- ☒ Non-incremental (i.e., batch method)
- ☒ Greedy algorithm (can result in suboptimal accuracy)

CART Algorithm (Classification and Regression Trees)

- The **CART (Classification and Regression Trees)** algorithm is a type of decision tree algorithm that can be used for both **classification** and **regression** tasks.
- For classification, CART uses the **Gini Index** as the splitting criterion, while for regression, it uses **Mean Squared Error (MSE)** to create splits.
- Like ID3, it is a greedy algorithm selecting the best attribute to split at every stage
- The CART approach restricts the splits to binary values for both categorical and continuous valued attributes.
- Thus, CART is a binary tree.

CART Algorithm (Classification and Regression Trees)

In this explanation, we will focus on the **CART algorithm for classification** and walk through an example using the **Gini Index**.

Steps of the CART Algorithm for Classification:

1. **Start with the Entire Dataset:**
   - The entire dataset becomes the root node.
2. **Calculate the Gini Index:**
   - For each feature, the algorithm evaluates different split points by calculating the **Gini Index** of the child nodes after the split.
   - The **Gini Index** for a node N is calculated as:

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

3. **Evaluate All Possible Splits**:
    - For each feature, the algorithm evaluates all possible split points and calculates the weighted **Gini Index** of the resulting child nodes.
4. **Choose the Best Split**:
    - Select the split that minimizes the weighted **Gini Index**.
5. **Recursively Repeat the Process**:
    - Repeat the process for each child node until a stopping criterion is met (such as the maximum depth of the tree, or the minimum number of samples per node).
6. **Prune the Tree (if necessary)**:
    - To prevent overfitting, CART can prune the tree by removing branches that contribute little to predictive power.
7. **Make Predictions**:
    - For classification, the predicted class at a leaf node is determined by the majority class of the instances in that node

Example of the CART Algorithm for Classification

Let's work through an example to build a **CART decision tree** for a simple dataset.

ID	Age	Income	Buys Product?
1	25	50K	No
2	35	60K	No
3	45	80K	Yes
4	50	90K	Yes
5	23	40K	No
6	40	70K	Yes
7	60	100K	Yes
8	48	85K	Yes

ID	Age	Income	Buys Product?
1	25	50K	No
2	35	60K	No
3	45	80K	Yes
4	50	90K	Yes
5	23	40K	No
6	40	70K	Yes
7	60	100K	Yes
8	48	85K	Yes

Step 1: Calculate the Gini Index for the Root Node

We start with the entire dataset as the root node. There are **8 instances**:

- 5 instances where **Buys Product = Yes**.
- 3 instances where **Buys Product = No**.

The Gini Index for the root node is calculated as:

$$Gini(D) = 1 - \left(\frac{5}{8}\right)^2 - \left(\frac{3}{8}\right)^2 = 1 - 0.390625 - 0.140625 = 0.46875$$

ID	Age	Income	Buys Product?
1	25	50K	No
2	35	60K	No
3	45	80K	Yes
4	50	90K	Yes
5	23	40K	No
6	40	70K	Yes
7	60	100K	Yes
8	48	85K	Yes

Step 2: Evaluate Possible Splits

Now, let's evaluate possible splits for the **Age** and **Income** features.

Split on **Age**:

We will check a potential split where **Age ≤ 35**:

- Left Child (Age ≤ 35): Instances 1, 2, 5 → 0 Yes, 3 No.

$$Gini(\text{Left Child}) = 1 - \left(\frac{0}{3}\right)^2 - \left(\frac{3}{3}\right)^2 = 1 - 0 - 1 = 0$$

- Right Child (Age > 35): Instances 3, 4, 6, 7, 8 → 5 Yes, 0 No.

$$Gini(\text{Right Child}) = 1 - \left(\frac{5}{5}\right)^2 - \left(\frac{0}{5}\right)^2 = 1 - 1 - 0 = 0$$

The weighted Gini Index for this split is:

$$Gini(Age) = \frac{3}{8} \times 0 + \frac{5}{8} \times 0 = 0$$

ID	Age	Income	Buys Product?
1	25	50K	No
2	35	60K	No
3	45	80K	Yes
4	50	90K	Yes
5	23	40K	No
6	40	70K	Yes
7	60	100K	Yes
8	48	85K	Yes

Split on **Income**:

Now, let's consider a potential split on **Income ≤ 70K**:

- Left Child (Income ≤ 70K): Instances 1, 2, 5, 6 → 1 Yes, 3 No.

$$Gini(\text{Left Child}) = 1 - \left(\frac{1}{4}\right)^2 - \left(\frac{3}{4}\right)^2 = 1 - 0.0625 - 0.5625 = 0.375$$

- Right Child (Income > 70K): Instances 3, 4, 7, 8 → 4 Yes, 0 No.

$$Gini(\text{Right Child}) = 1 - \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2 = 1 - 1 - 0 = 0$$

The weighted Gini Index for this split is:

$$Gini(Income) = \frac{4}{8} \times 0.375 + \frac{4}{8} \times 0 = 0.1875$$

Step 3: Choose the Best Split

- The Gini Index for the **Age** split is 0.
- The Gini Index for the **Income** split is 0.1875.

Since the split on **Age ≤ 35** results in a Gini Index of 0 (perfect purity), we choose it as the best split.

Step 4: Split the Data Based on Age

After splitting on **Age ≤ 35**, we get two child nodes:

1. **Left Child (Age ≤ 35)**: Instances 1, 2, 5 → All are "No" (pure node).
2. **Right Child (Age > 35)**: Instances 3, 4, 6, 7, 8 → All are "Yes" (pure nod

Both nodes are **pure** (Gini = 0), so no further splits are needed.

ID	Age	Income	Buys Product?
1	25	50K	No
2	35	60K	No
3	45	80K	Yes
4	50	90K	Yes
5	23	40K	No
6	40	70K	Yes
7	60	100K	Yes
8	48	85K	Yes

Step 5: Final Decision Tree

The decision tree looks like this:

```
        Age <= 35
        /       \
      No         Yes
```

Step 6: Predictions Using the Tree

- If Age ≤ 35, predict **No**.
- If Age > 35, predict **Yes**.

Example Predictions:

1. A person with **Age = 30** and **Income = 50K** → Prediction: **No** (because Age ≤ 35).
2. A person with **Age = 50** and **Income = 90K** → Prediction: **Yes** (because Age > 35).

Advantages of CART:

- Results are simple to understand.
- Classification and regression trees are Nonparametric and Nonlinear.
- Classification and regression trees implicitly perform feature selection.
- Outliers have no meaningful effect on CART.
- It requires minimal supervision and produces easy-to-understand models

Occam's Razor is a philosophical and scientific principle that suggests the simplest explanation or solution is usually the best one.

Occam's Razor promotes favoring simplicity over complexity in explanation and theory, unless evidence strongly supports the more complex option.

Feature	ID3 (Iterative Dichotomiser 3)	CART (Classification and Regression Trees)	C4.5
Inventor	Ross Quinlan	Leo Breiman and others	Ross Quinlan
Type	Classification	Both Classification and Regression	Classification
Splitting Criterion	Information Gain	Gini Index (for classification), Variance Reduction (for regression)	Gain Ratio (an enhancement of Information Gain)
Handling of Continuous Data	Does not handle continuous data directly; requires discretization	Handles continuous data by splitting the range	Handles continuous data by creating threshold-based splits
Pruning	No pruning (prone to overfitting)	Pruning included (cost complexity pruning)	Includes post-pruning to reduce overfitting

Tree Structure	Multibranch trees	Binary trees only	Multibranch trees
Missing Values	Does not handle missing values	Can handle missing values	Can handle missing values
Output	Discrete class labels	Discrete class labels (for classification) or numeric values (for regression)	Discrete class labels
Performance	Faster but prone to overfitting	More flexible and robust due to pruning and binary splits	Improved performance over ID3 due to gain ratio and pruning
Usage	Simple tasks with categorical data	Tasks requiring flexibility (both classification and gression)	Classification tasks with improved handling of continuous and missing data

Linear Regression

- **Linear Regression** is a type of supervised Machine Learning. It is used to solve regression problems
- **Linear Regression** is a statistical method for modeling the relationship between a **dependent variable** (also called the target or output) and one or more **independent variables** (also called features or predictors).
- It assumes that the relationship between the dependent variable and the independent variables is linear.
- Linear regression can be classified into two types:

    1.Simple Linear Regression (with one independent variable)

    2.Multiple Linear Regression (with more than one independent variable)

Simple Linear Regression: Example

Suppose we have data representing a company's advertising expenses (x) and the resulting sales (y) over 5 months:

Advertising Expense (x)	Sales (y)
2	4
3	5
5	7
7	10
9	15

Multiple Linear Regression

area	bedrooms	bathrooms	guestroom	basement	hotwater heating	aircondit ioning	parking	price
7420	4	2	no	no	no	yes	2	13300000
8960	4	4	no	no	no	yes	3	12250000
9960	3	2	no	yes	no	no	2	12250000
7500	4	2	no	yes	no	yes	3	12215000
7420	4	1	yes	yes	no	yes	2	11410000
7500	3	3	no	yes	no	yes	2	10850000
8580	4	3	no	no	no	yes	2	10150000
16200	5	3	no	no	no	no	0	10150000
8100	4	1	yes	yes	no	yes	2	9870000
5750	3	2	yes	no	no	yes	1	9800000

House Price prediction Dataset

Simple Linear Regression

1. **Simple Linear Regression**: Involves only one independent variable.
   - Model: $y = \beta_0 + \beta_1 x + \epsilon$
2. **Multiple Linear Regression**: Involves two or more independent variables.
   - Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n + \epsilon$

Where:

- y is the predicted (dependent) variable.
- $x_1, x_2, \ldots, x_n$ are the independent (explanatory) variables.
- β_0 is the intercept (constant term).
- $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients (slopes) of the independent variables.
- ϵ is the error term, capturing the discrepancy between the observed and predicted values.

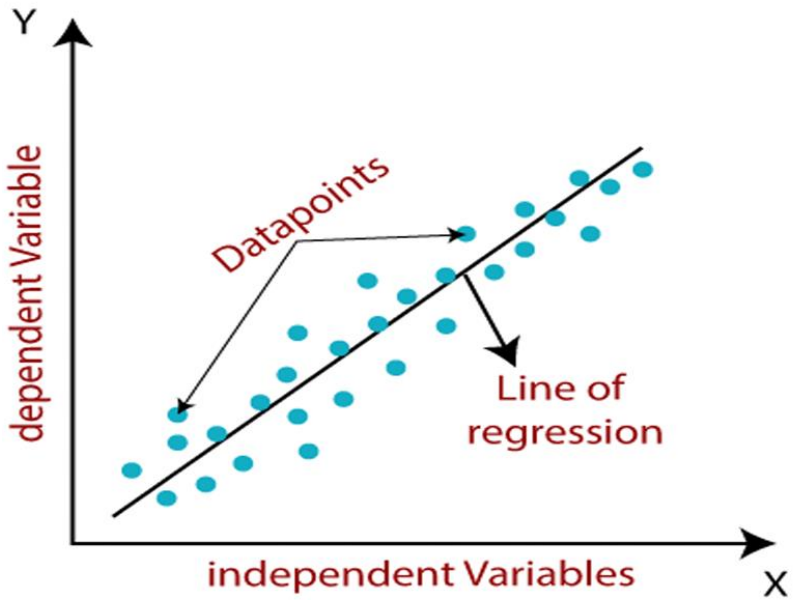
Objective of Linear Regression

- The goal of linear regression is to find the best-fitting line (in the case of simple regression) or hyperplane (in the case of multiple regression) that minimizes the error between predicted values and actual values.
- This is typically done by minimizing the **sum of squared errors (SSE)** also known as the **cost function**

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

where:

- y_i is the actual value of the dependent variable for the i-th observation.
- $\hat{y}_i = \beta_0 + \beta_1 x_i$ is the predicted value of y for the i-th observation.



Steps in Linear Regression:

1. **Data Preparation**: We need a dataset with one or more independent variables and a dependent variable.
2. **Model Representation**: Represent the relationship between variables using a linear equation.
3. **Parameter Estimation**: Find the coefficients ($\beta_0, \beta_1, \ldots$) that minimize the error between predicted and actual values.
4. **Model Evaluation**: Evaluate the model using metrics like MSE, R^2, and others.
5. **Prediction**: Use the model to make predictions on new data.

Simple Linear Regression: Example

Suppose we have data representing a company's advertising expenses (x) and the resulting sales (y) over 5 months:

Advertising Expense (x)	Sales (y)
2	4
3	5
5	7
7	10
9	15

Simple Linear Regression: Solution

Calculating the Coefficients

Using the least squares approach, the formulas for the slope (β_1) and intercept (β_0) are:

$$\beta_1 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\beta_0 = \bar{y} - \beta_1 \bar{x}$$

where:

- $\bar{x}$ and $\bar{y}$ are the means of the x and y values, respectively.

Where:

- x_i and y_i are the individual values of the independent and dependent variables, respectively.
- n is the number of observations.

Simple Linear Regression: Solution

1. **Calculate $\bar{x}$ and $\bar{y}$:**

$$\bar{x} = \frac{2+3+5+7+9}{5} = 5.2$$

$$\bar{y} = \frac{4+5+7+10+15}{5} = 8.2$$

Advertising Expense (x)	Sales (y)
2	4
3	5
5	7
7	10
9	15

2. **Calculate β_1:**

$$\beta_1 = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

After substituting values, we get:

$$\beta_1 = \frac{(2-5.2)(4-8.2) + (3-5.2)(5-8.2) + (5-5.2)(7-8.2) + (7-5.2)(10-8.2) + (9-5.2)(15-8.2)}{(2-5.2)^2 + (3-5.2)^2 + (5-5.2)^2 + (7-5.2)^2 + (9-5.2)^2}$$

Calculating each term, we find $\beta_1 \approx 1.43$.

3. **Calculate β_0:**

$$\beta_0 = \bar{y} - \beta_1 \bar{x} = 8.2 - 1.43 \times 5.2 \approx 0.66$$

4. **Final Regression Equation:** The estimated regression line is:

$$\hat{y} = 0.66 + 1.43x$$

Simple Linear Regression: Solution

Using the Model to Make Predictions

If we want to predict sales when advertising expenses (x) are 6, we substitute $x = 6$ into the regression equation:

$$\hat{y} = 0.66 + 1.43 \times 6 \approx 9.24$$

So, the predicted sales would be approximately 9.24 units.

Simple Linear Regression: Solution

Model Evaluation

To evaluate the model, we use metrics like:

1. **Mean Squared Error (MSE)**:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

This measures the average squared difference between the observed actual outcomes and the predicted outcomes.

2. R^2 **(Coefficient of Determination)**:

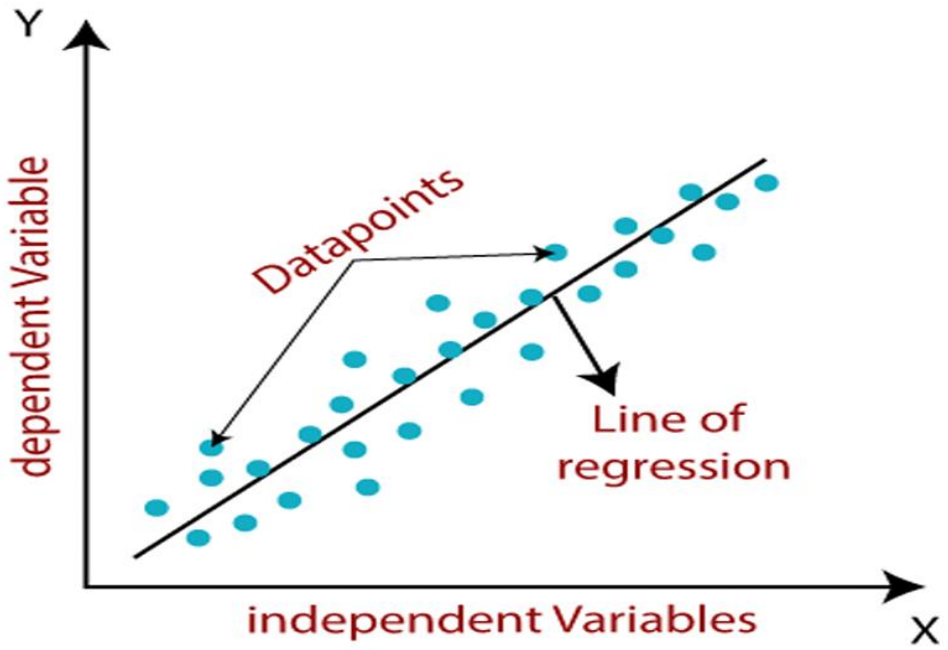
$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

This measures how well the independent variable explains the variability in the dependent variable. R^2 ranges from 0 to 1, where 1 means perfect prediction.

For our small dataset, these metrics would indicate how well the model fits the training data.

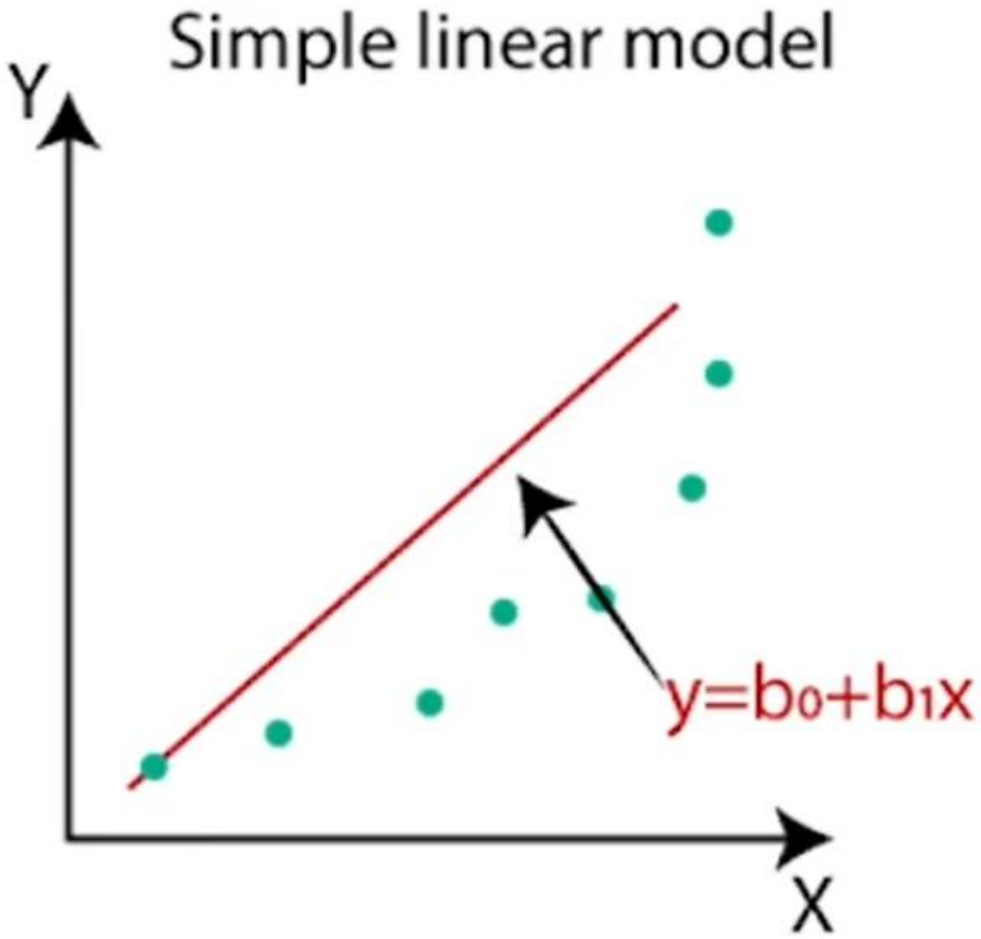
Simple Linear Regression

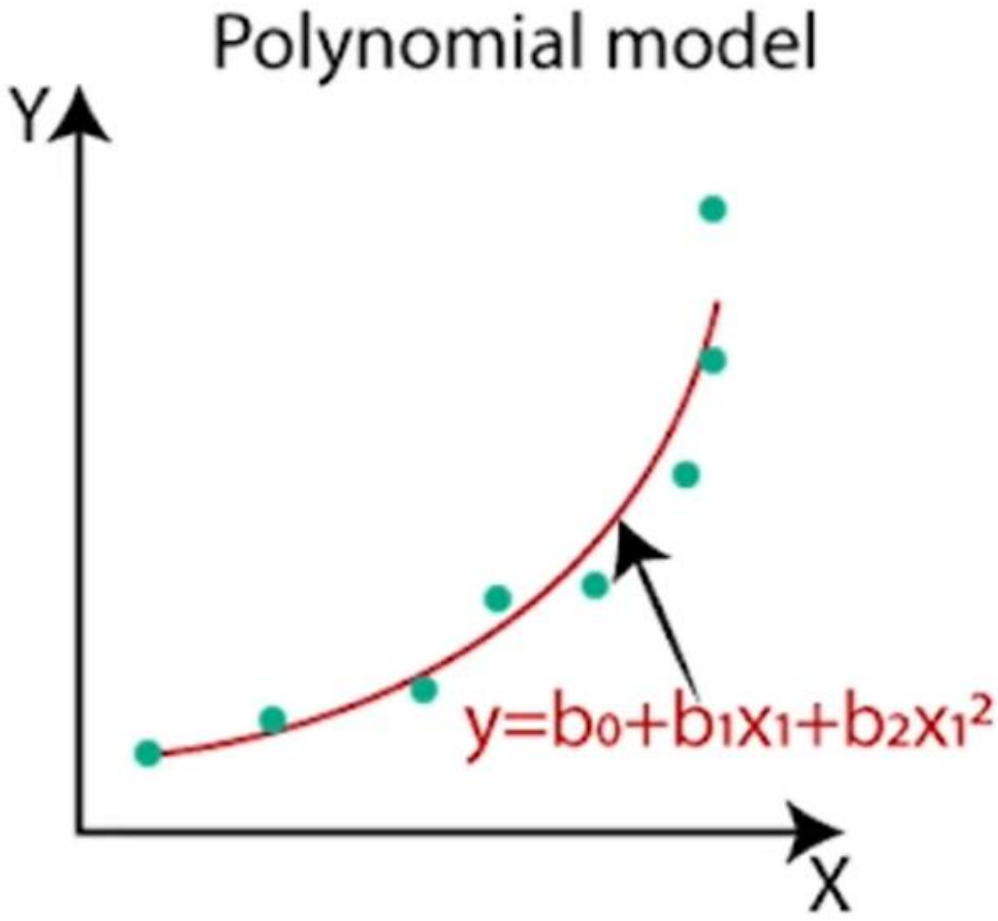
The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



- It works well for linear data only ,not good for non-linear data
- The main drawback of Linear Regression is ,it was not given good performance to non-linear data

Simple Linear Regression





Techniques to handle non-linear data

Two common techniques to handle non-linear relationships in data within the framework of linear regression

1. **Transformations**
2. **polynomial regression**

- These approaches allow us to use linear modeling methods even when the underlying relationship between variables is non-linear.
- Here's a closer look at how they work and why they can effectively address non-linear regression problems:

1. Transformations

Transformations apply mathematical functions to the data, especially the independent variables, to linearize the relationship. By doing so, you can model non-linear data with standard linear regression techniques.

Common Types of Transformations

- **Logarithmic Transformation**:
  - Useful when the data shows an exponential growth pattern, like income over time or population growth.
  - Converts an exponential relationship $y = a \cdot e^{b \cdot x}$ into a linear form: $\ln(y) = \ln(a) + b \cdot x$.
- **Square Root Transformation**:
  - Effective when data has a right-skewed distribution or when variance increases with the value of the independent variable.
  - This transformation can stabilize variance and improve the linearity of the relationship.

2. Polynomial Regression

- Polynomial regression addresses non-linearity by adding polynomial terms of the independent variable(s), creating a curve-fitting model.
- Instead of transforming the entire variable, it raises each variable to various powers, effectively adding flexibility to the model.

How Polynomial Regression Works

- **Model Structure**:
  - For a single variable x, a polynomial regression model of degree n looks like:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_n x^n + \epsilon$$

  - Each term x^i (where i is the power) represents an additional feature, allowing the model to form curves and capture non-linear trends.
- **Choosing the Polynomial Degree**:
  - A low-degree polynomial (like a quadratic or cubic) often captures basic non-linear patterns effectively.
  - Higher degrees provide more flexibility but risk overfitting, where the model captures noise rather than the true pattern in the data.
  - Cross-validation is typically used to identify the optimal degree for balancing fit and generalization.

Polynomial Regression Equation with Two Independent Features

For two independent variables, x_1 and x_2, a degree-2 polynomial regression equation is:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2 + \epsilon$$

- Here’s a detailed solution to polynomial regression with an example of a degree-2 polynomial (also known as **quadratic regression**).
- This approach will cover how to set up the model, interpret coefficients, and make predictions.

Problem Statement

- Suppose we have data on the speed of a car (xxx) and the corresponding stopping distance (yyy).
- The relationship between speed and stopping distance is non-linear, as higher speeds increase stopping distance exponentially rather than linearly.
- We’ll use polynomial regression to model this relationship.

Speed (x)	Stopping Distance (y)
10	9
20	30
30	60
40	100
50	150

Step 1: Define the Polynomial Regression Model

For a degree-2 polynomial, the model looks like this:

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \epsilon$$

where:

- y: dependent variable (stopping distance),
- x: independent variable (speed),
- β_0: intercept term,
- β_1: linear coefficient,
- β_2: quadratic coefficient, representing the effect of x^2 on y,
- ϵ: error term.

Step 2: Expand the Features

To apply polynomial regression, we need to create additional features by raising x to the power of 2:

1. Let x represent the speed.
2. Create a new feature x^2, which is the square of the speed.

This transforms our single feature x into two features: x and x^2.

Step 3: Set Up the Model with Matrix Notation

Let's assume we have a dataset of n observations with:

- X: an $n \times 2$ matrix, where the first column is x (the speed values) and the second column is x^2 .
- y: a column vector of n stopping distances.

The model in matrix form can be written as:

$$\mathbf{y} = \mathbf{X} \cdot \beta + \epsilon$$

where:

- $\mathbf{y}$: vector of y values (stopping distances),
- $\mathbf{X}$: matrix with columns $[1, x, x^2]$ (first column is all ones for β_0),
- β: vector of coefficients $[\beta_0, \beta_1, \beta_2]$,

Step 4: Estimate the Coefficients

We estimate the coefficients using the **ordinary least squares** (OLS) method. In OLS, the coefficients β are calculated to minimize the sum of squared errors:

$$\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

This formula provides the best-fit values for β_0, β_1, and β_2.

Step 5: Apply the Model to Data (Example)

Assume we have the following example data for speed and stopping distance:

Speed (x)	Stopping Distance (y)
10	9
20	30
30	60
40	100
50	150

Step 5a: Create Polynomial Features

Create x^2 values for each speed value:

Speed (x)	x^2	Stopping Distance (y)
10	100	9
20	400	30
30	900	60
40	1600	100
50	2500	150

Let $\mathbf{X}$ be the matrix of features with a column of ones for the intercept:

$$\mathbf{X} = \begin{bmatrix} 1 & 10 & 100 \\ 1 & 20 & 400 \\ 1 & 30 & 900 \\ 1 & 40 & 1600 \\ 1 & 50 & 2500 \end{bmatrix}$$

And $\mathbf{y}$ is the vector of stopping distances:

$$\mathbf{y} = \begin{bmatrix} 9 \\ 30 \\ 60 \\ 100 \\ 150 \end{bmatrix}$$

Step 5c: Calculate the Coefficients

Using the formula $\beta = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, we solve for β_0, β_1, and β_2.

Let's assume after computation, we get:

$$\beta_0 = 5, \quad \beta_1 = -1.2, \quad \beta_2 = 0.1$$

Thus, the model is:

$$y = 5 - 1.2x + 0.1x^2$$

Use the Model for Predictions

To predict the stopping distance at a speed of $x = 35$:

1. Substitute $x = 35$ into the model:

$$y = 5 - 1.2(35) + 0.1(35^2)$$

2. Calculate each term:
   - $-1.2 \times 35 = -42$
   - $0.1 \times 35^2 = 122.5$
   - Sum: $y = 5 - 42 + 122.5 = 85.5$

So, the predicted stopping distance at 35 units of speed is approximately 85.5 units.

Polynomial Regression

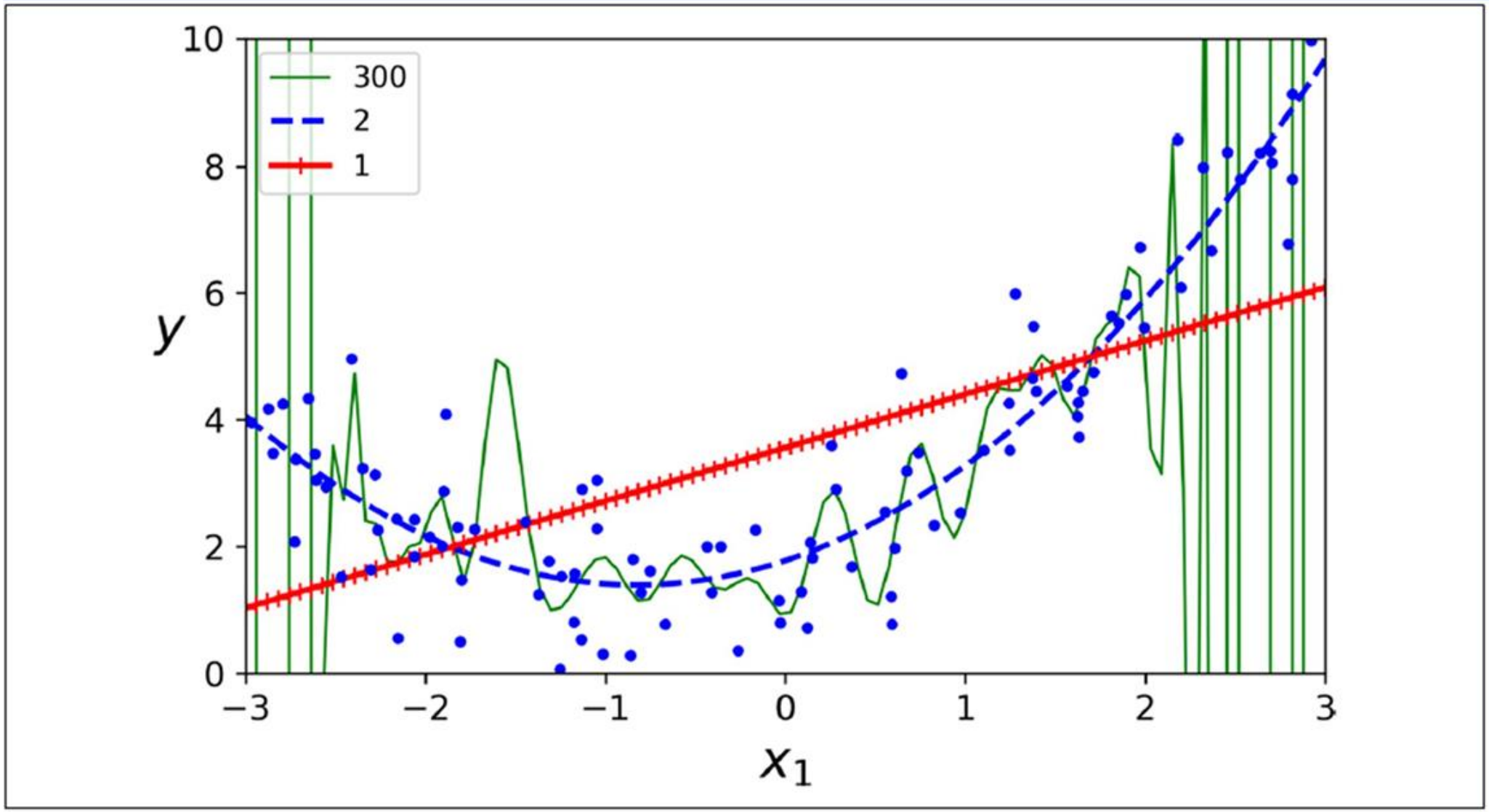


Figure 4-14. High-degree Polynomial Regression

- The high-degree Polynomial Regression model is severely overfitting the training data, while the linear model is underfitting it.
- The model that will generalize best in this case is the quadratic model.
- How do we decide how complex our model should be?
- How do we tell that our model is overfitting or underfitting the data?
  - Cross Validation!
  - Learning curves

Learning Curves and CV

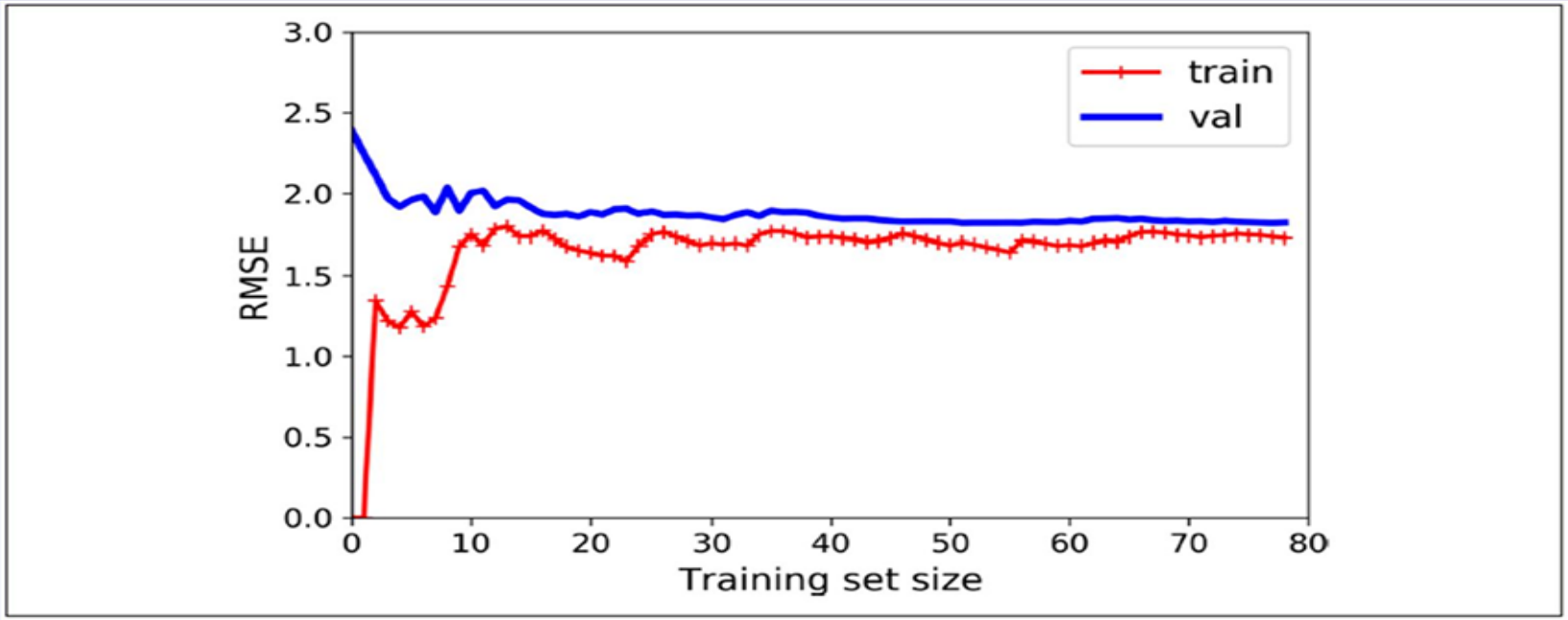


Figure 4-15. Learning curves - linear regression

- These learning curves are **typical of a model that's underfitting**. Both curves have reached a plateau; they are close and fairly high.
  - adding more training examples will not help.

Learning Curves and CV

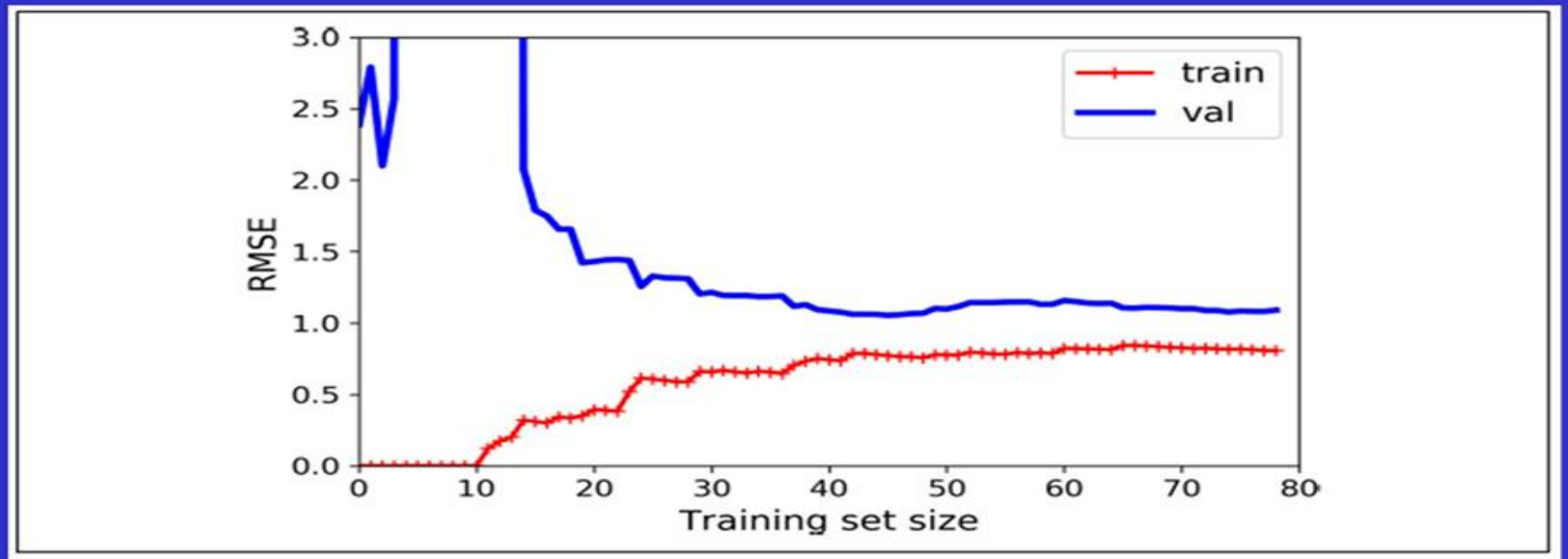


Figure 4-16. Learning curves for the 10th-degree polynomial model

- The error on the training data is much lower than with the Linear Regression model.
- There is a gap between the curves - a hallmark of an **overfitting** model.
- Larger the training set, closer the two curves.

Bias / Variance Tradeoff

- A model's generalization error can be expressed as the sum of three very different errors: **Bias, Variance** and **Irreducible error.**

Bias / Variance Tradeoff

1. Bias

- **Definition**: Bias is the error introduced by approximating a real-world problem, which may be complex, with a simplified model. In other words, it's the error due to the assumptions made by the model to simplify the learning process.
- **Cause**: Bias arises when the model is too simple to capture the underlying patterns in the data. For example, a linear model may introduce high bias if the true relationship is non-linear.
- **Effect**: High bias leads to **underfitting**, where the model does not learn the data patterns well enough and performs poorly on both the training and test datasets.

Bias / Variance Tradeoff

2. **Variance**

- **Definition**: Variance is the error introduced by the model's sensitivity to small fluctuations in the training set. A model with high variance learns the noise in the training data, making it overly complex.
- **Cause**: High variance is often the result of using a model that is too complex for the given dataset, allowing it to fit even random noise in the training data.
- **Effect**: High variance leads to **overfitting**, where the model performs well on the training data but poorly on unseen test data because it has "memorized" the training data rather than learning the underlying patterns.

Bias / Variance Tradeoff

3. Irreducible Error

- **Definition**: Irreducible error represents the noise or randomness in the data itself, which cannot be eliminated by any model. This could be due to measurement errors, inherent variability in the system being modeled, or other unknown factors.
- **Cause**: Irreducible error is inherent to the data and is independent of the model used. It exists because no model can account for every possible variable or random effect.
- **Effect**: Even with the best possible model, this component of the error remains and represents the limit of a model's accuracy.

The goal in model building is to achieve a balance between bias and variance to minimize the overall error (generalization error), recognizing that the irreducible error is unavoidable. This trade-off is often referred to as the **bias-variance trade-off**.

Aspect	Transformations	Polynomial Regression
Approach	Apply mathematical functions to data	Add polynomial terms to fit curves
Purpose	Linearize data with a non-linear pattern	Capture curved relationships directly
When to Use	When there is an identifiable transformation (log, square root, etc.) that fits the data	When data has no single transformation that fits well
Pros	Reduces need for high-order polynomials, interpretable results	Captures complex non-linear trends effectively
Cons	Requires selection of appropriate transformation	Higher degrees increase risk of overfitting

Linear vs. Polynomial regression

Feature	Linear Regression	Polynomial Regression
Assumption	The relationship between the dependent and independent variables is linear.	The relationship between the dependent and independent variables is non-linear.
Model	A linear equation is used to fit the data.	A polynomial equation is used to fit the data.
Complexity	Simpler model.	More complex model.
Accuracy	May not be accurate for non-linear data.	Can be more accurate for non-linear data.
Overfitting	Less prone to overfitting.	More prone to overfitting.
Outliers	Less sensitive to outliers.	Sensitive to outliers.

Performance Measures for regression Algorithms

1. Mean Absolute Error (MAE)

MAE calculates the average absolute difference between predicted and actual values. It's a straightforward way to see how far off predictions are, on average.

Advertising Expense (x)	Sales (y)
2	4
3	5
5	7
7	10
9	15

Formula:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

Final Regression Equation: The estimated regression line is:

$$\hat{y} = 0.66 + 1.43x$$

Example:

If actual values are $[10, 20, 30]$ and predictions are $[12, 18, 29]$:

$$\text{MAE} = \frac{|10-12| + |20-18| + |30-29|}{3} = \frac{2+2+1}{3} = 1.67$$

Performance Measures for regression Algorithms

2. Mean Squared Error (MSE)

MSE measures the average squared difference between predicted and actual values. Squaring emphasizes larger errors, which can be helpful if we want to penalize big mistakes.

Formula:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Example:

If actual values are $[10, 20, 30]$ and predictions are $[12, 18, 29]$:

$$\text{MSE} = \frac{(10-12)^2 + (20-18)^2 + (30-29)^2}{3} = \frac{4+4+1}{3} = 3.0$$

Performance Measures for regression Algorithms

3. Root Mean Squared Error (RMSE)

RMSE is the square root of MSE, which brings the error back to the same unit as the target variable. It penalizes large errors more than small ones but is often more interpretable than MSE.

Formula:

$$\text{RMSE} = \sqrt{\text{MSE}}$$

Example:

Using the MSE from the previous example:

$$\text{RMSE} = \sqrt{3.0} \approx 1.73$$

Performance Measures for Regression Algorithms

4. R-squared (R^2)

R^2 indicates the proportion of variance in the dependent variable that is predictable from the independent variable(s). It ranges from 0 to 1, where 1 means perfect prediction.

Formula:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}$$

where $\bar{y}$ is the mean of the actual values.

Example:

If actual values are $[10, 20, 30]$ with a mean of 20, and predictions are $[12, 18, 29]$:

1. Residual Sum of Squares (RSS): $(10 - 12)^2 + (20 - 18)^2 + (30 - 29)^2 = 9$
2. Total Sum of Squares (TSS): $(10 - 20)^2 + (20 - 20)^2 + (30 - 20)^2 = 200$
3. $R^2 = 1 - \frac{9}{200} = 0.955$

An R^2 of 0.955 indicates that about 95.5% of the variance is explained by the model.

5. Mean Absolute Percentage Error (MAPE)

MAPE represents the average percentage error between actual and predicted values. It is particularly useful for interpreting error in a percentage form.

Formula:

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Example:

If actual values are $[10, 20, 30]$ and predictions are $[12, 18, 29]$:

$$\text{MAPE} = \frac{100\%}{3} \left(\left| \frac{10 - 12}{10} \right| + \left| \frac{20 - 18}{20} \right| + \left| \frac{30 - 29}{30} \right| \right) = \frac{100\%}{3} (0.2 + 0.1 + 0.033) \approx 11.1\%$$

These accuracy measures offer diverse ways to assess regression model performance based on the importance of large vs. small errors, units of error, and interpretability needs.

Performance Measures for Classification Algorithms

1. Confusion Matrix

A **Confusion Matrix** provides a summary of the prediction results for a classification problem. It compares predicted vs. actual values across classes, giving insights into true positives, false positives, false negatives, and true negatives.

Example

Consider a binary classification problem with **Class A (Positive)** and **Class B (Negative)**. The confusion matrix might look like this:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positives (TP)	False Negatives (FN)
Actual Negative	False Positives (FP)	True Negatives (TN)

Performance Measures for Classification Algorithms

- True Positive (TP): The model correctly predicts the positive class.
- False Positive (FP): The model incorrectly predicts the positive class when it's actually negative.
- True Negative (TN): The model correctly predicts the negative class.
- False Negative (FN): The model incorrectly predicts the negative class when it's actually positive.
- Sensitivity (True Positive Rate): The proportion of actual positive samples correctly predicted.
- Specificity (True Negative Rate): The proportion of actual negative samples correctly predicted.

Performance Measures for Classification Algorithms

Metric
Accuracy
Precision
Recall
F1 Score
Specificity
AUC (hypothetical)
Log Loss
MCC
Balanced Accuracy

Performance Measures for Classification Algorithms

	Predict Positive	Predict Negative
Actual Positive	**TP=50**	**FN=10**
Actual Negative	**FP=5**	**TN=100**

2. Accuracy

Accuracy is the proportion of correct predictions out of all predictions made.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Example Calculation

Using the values from our confusion matrix:

$$\text{Accuracy} = \frac{50 + 100}{50 + 100 + 5 + 10} = \frac{150}{165} \approx 0.91$$

So, **accuracy is 91%**. However, accuracy can be misleading, especially for imbalanced datasets.

Performance Measures for Classification Algorithms

3. Precision

Precision measures the proportion of correctly predicted positive observations out of all positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP}$$

	Predict Positive	Predict Negative
Actual Positive	**TP=50**	**FN=10**
Actual Negative	**FP=5**	**TN=100**

Example Calculation

$$\text{Precision} = \frac{50}{50 + 5} = \frac{50}{55} \approx 0.91$$

So, **precision is 91%.**

Performance Measures for Classification Algorithms

	Predict Positive	Predict Negative
Actual Positive	TP=50	FN=10
Actual Negative	FP=5	TN=100

4. Recall (Sensitivity or True Positive Rate)

Recall measures the proportion of actual positives correctly identified by the model.

$$\text{Recall} = \frac{TP}{TP + FN}$$

Example Calculation

$$\text{Recall} = \frac{50}{50 + 10} = \frac{50}{60} \approx 0.83$$

So, **recall is 83%.**

$$\textit{True positive rate (TPR)} = \frac{\textit{Positives cases surfaced}}{\textit{Total number of positive cases in the data set}}$$

Performance Measures for Classification Algorithms

	Predict Positive	Predict Negative
Actual Positive	**TP=50**	**FN=10**
Actual Negative	**FP=5**	**TN=100**

5. F1 Score

The **F1 Score** is the harmonic mean of precision and recall. It balances the two, which is useful if you want to balance precision and recall.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Example Calculation

With Precision = 0.91 and Recall = 0.83:

$$\text{F1 Score} = 2 \times \frac{0.91 \times 0.83}{0.91 + 0.83} \approx 0.87$$

So, **F1 Score is 87%.**

Performance Measures for Classification Algorithms

6. Specificity (True Negative Rate)

Specificity is the proportion of actual negatives correctly identified by the model.

$$\text{Specificity} = \frac{TN}{TN + FP}$$

	Predict Positive	Predict Negative
Actual Positive	TP=50	FN=10
Actual Negative	FP=5	TN=100

Example Calculation

$$\text{Specificity} = \frac{100}{100 + 5} = \frac{100}{105} \approx 0.95$$

So, specificity is 95%.

$$\textit{False positive rate (FPR)} = \frac{\textit{Negative cases surfaced}}{\textit{Total number of negative cases in the data set}}$$

Performance Measures for Classification Algorithms

7. ROC Curve and AUC (Area Under Curve)

The **ROC (Receiver Operating Characteristic) Curve** is a graphical plot illustrating the performance of a binary classifier as its discrimination threshold is varied. It plots the **True Positive Rate** (Recall) against the **False Positive Rate** (1 - Specificity).

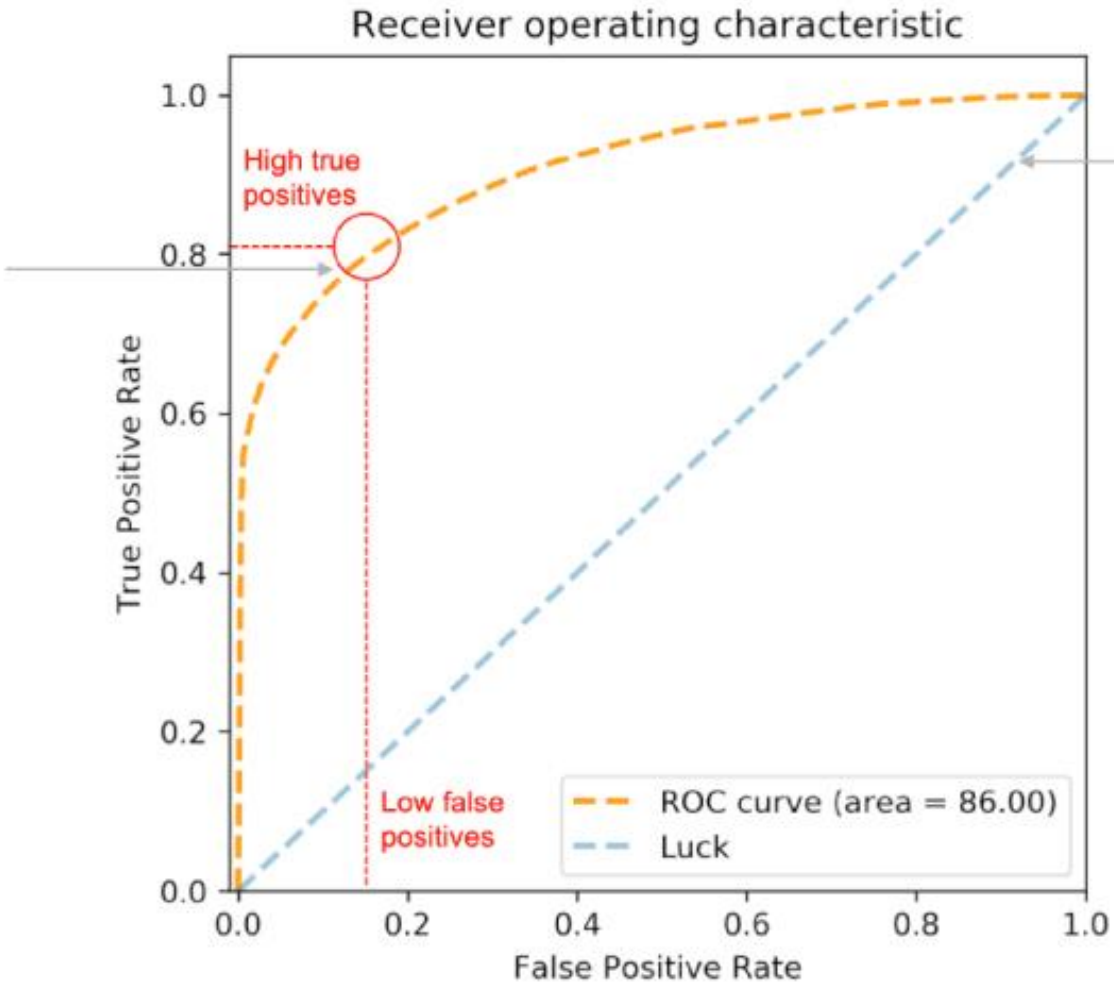
- **AUC** (Area Under the Curve) represents the area under the ROC curve. An AUC of 1.0 represents a perfect model, while an AUC of 0.5 indicates a random model.

Example Calculation

To generate an ROC curve, calculate the TPR and FPR at various threshold values and plot the curve.

Performance Measures for Classification Algorithms

An ideal classifier can predict failures with high true positives with low false positives



This straight line represents a predictive model that is "randomly guessing"

Performance Measures for Classification Algorithms

8. Logarithmic Loss (Log Loss)

Log Loss measures the performance of a classifier whose output is a probability value between 0 and 1. The formula for binary classification is:

$$\text{Log Loss} = -\frac{1}{N}\sum_{i=1}^{N}[y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

where:

- y_i is the actual label (0 or 1),
- p_i is the predicted probability of the positive class.

Example Calculation

If a sample's true class label is 1, and the model predicts a probability of 0.9 for this class, the log loss for this prediction is:

$$-(1 \cdot \log(0.9) + (1 - 1) \cdot \log(1 - 0.9)) \approx 0.105$$

Performance Measures for Classification Algorithms

9. Matthews Correlation Coefficient (MCC)

MCC measures the correlation between observed and predicted classifications, ranging from -1 (total disagreement) to 1 (perfect prediction).

$$\text{MCC} = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

Example Calculation

Using our confusion matrix values:

$$\text{MCC} = \frac{(50 \cdot 100) - (5 \cdot 10)}{\sqrt{(50 + 5)(50 + 10)(100 + 5)(100 + 10)}} \approx 0.83$$

	Predict Positive	Predict Negative
Actual Positive	**TP=50**	**FN=10**
Actual Negative	**FP=5**	**TN=100**

Performance Measures for Classification Algorithms

10. Balanced Accuracy

For imbalanced datasets, **Balanced Accuracy** averages the recall obtained on each class. It's the arithmetic mean of sensitivity (Recall for Positive class) and specificity (Recall for Negative class).

$$\text{Balanced Accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$$

Example Calculation

Using our sensitivity (0.83) and specificity (0.95):

$$\text{Balanced Accuracy} = \frac{0.83 + 0.95}{2} = 0.89$$

	Predict Positive	Predict Negative
Actual Positive	**TP=50**	**FN=10**
Actual Negative	**FP=5**	**TN=100**

Summary Table of Metrics

Here's a summary table of all metrics using our example confusion matrix:

Metric	Value
Accuracy	0.91
Precision	0.91
Recall	0.83
F1 Score	0.87
Specificity	0.95
AUC (hypothetical)	~0.9
Log Loss	Depends on probabilities
MCC	0.83
Balanced Accuracy	0.89

The MNIST dataset is a well-known dataset in machine learning and computer vision, primarily used for training and testing image processing systems. It contains 60,000 training images and 10,000 testing images of handwritten digits (0 through 9), each sized 28x28 pixels. The dataset was derived from a larger dataset created by the National Institute of Standards and Technology (NIST), with modifications to make it more accessible and practical for machine learning research.

MNIST is widely used as a benchmark for evaluating algorithms in classification tasks, especially for models like neural networks and convolutional neural networks (CNNs).

1. **Regularization**

- **Purpose**: Regularization aims to reduce overfitting by adding a penalty to the model's complexity, thus simplifying the model.
- **How It Works**: In regression, regularization methods add a penalty term to the loss function, which depends on the size of the model parameters (coefficients). This added penalty discourages the model from learning overly complex patterns, which helps reduce variance and makes the model more generalizable.
- **Types**: Common regularization methods include **L2 regularization** (Ridge regression) and **L1 regularization** (Lasso regression), each with a different approach to penalizing model complexity.

2. **Ridge Regression (L2 Regularization)**

- **Definition**: Ridge regression, also known as L2 regularization, adds the sum of the squared values of the coefficients to the loss function. The objective is to minimize the sum of squared residuals along with this penalty term.
- **Penalty Term**: The L2 penalty is proportional to the sum of the squares of the coefficients:

$$\text{Loss function} = \text{MSE} + \lambda \sum_{j=1}^{p} \beta_j^2$$

where λ is a hyperparameter controlling the amount of regularization, β_j are the coefficients, and MSE is the Mean Squared Error.

- **Effect**: Ridge regression shrinks the coefficients, but it does not set any to zero. This means that all features are included in the final model, but their impact is reduced.
- **Use Cases**: Ridge is useful when there are many features, and you want to prevent the model from being too complex, but don't want to eliminate any features entirely.

3. **Lasso Regression (L1 Regularization)**

- **Definition**: Lasso regression, or L1 regularization, adds the absolute values of the coefficients to the loss function. This encourages some coefficients to shrink exactly to zero, effectively performing feature selection.
- **Penalty Term**: The L1 penalty is proportional to the sum of the absolute values of the coefficients:

$$\text{Loss function} = \text{MSE} + \lambda \sum_{j=1}^{p} |\beta_j|$$

  where λ is a hyperparameter, and β_j are the coefficients.
- **Effect**: Lasso regression can shrink some coefficients to zero, effectively removing certain features from the model. This makes it a useful tool for feature selection, as it keeps only the most important predictors.
- **Use Cases**: Lasso is ideal when you have a high-dimensional dataset (many features) and suspect that only a subset of those features is truly predictive.

Elastic Net

Elastic Net is a regularization technique that combines both L1 (Lasso) and L2 (Ridge) penalties to overcome the limitations of using either regularization alone. It's especially useful when working with high-dimensional datasets, where there are many features and some may be highly correlated.

Key Concepts of Elastic Net

- **Combination of L1 and L2 Regularization**: Elastic Net applies both Lasso's L1 penalty (which encourages sparsity by pushing some coefficients to zero) and Ridge's L2 penalty (which helps shrink coefficients to manage multicollinearity).
- **Tunable Parameters**: It has two main tuning parameters:
  - **λ (lambda)**: Controls the overall strength of regularization.
  - **α (alpha)**: Controls the balance between L1 and L2 penalties.
    - When $\alpha = 1$: Elastic Net becomes equivalent to Lasso.
    - When $\alpha = 0$: Elastic Net behav[illegible]ke Ridge.

Formula

Elastic Net's cost function can be written as follows for a dataset with n features:

$$\text{Loss} = \text{MSE} + \lambda \left(\alpha \sum_{j=1}^{n} |\beta_j| + (1 - \alpha) \sum_{j=1}^{n} \beta_j^2 \right)$$

where:

- MSE is the Mean Squared Error of the model,
- β_j represents the coefficients,
- λ is the regularization strength,
- α adjusts the relative contributions of L1 and L2 penalties.

How Elastic Net Works

1. **Balance between L1 and L2 Regularization**: By blending L1 and L2 regularization, Elastic Net benefits from the strengths of both.
    - The L1 penalty (Lasso) helps with feature selection by shrinking some coefficients to zero.
    - The L2 penalty (Ridge) helps manage multicollinearity, which is especially important when features are highly correlated.
2. **Feature Selection and Multicollinearity**: In datasets with correlated features, Lasso may randomly select one feature and ignore others due to its nature of shrinking coefficients to zero. Elastic Net, however, can retain correlated features by applying both types of regularization.

Example Usage of Elastic Net

Suppose you are modeling a dataset with 100 features, and many of these features are correlated. Elastic Net would allow you to control for:

- Feature selection, by penalizing with L1 (some coefficients will be zero),
- Multicollinearity, by shrinking correlated features together rather than dropping them randomly.

Comparison: Ridge vs. Lasso

Feature	Ridge Regression (L2)	Lasso Regression (L1)
Penalty Term	$\lambda \sum_{j=1}^{p} \beta_j^2$	(\lambda \sum_{j=1}^{p}
Effect on Coefficients	Shrinks coefficients, but none go to zero	Can shrink coefficients to zero
Feature Selection	Does not perform feature selection	Performs feature selection
Use Case	When all features are expected to be relevant	When only a subset of features are relevant
Type of Regularization	L2 regularization	L1 regularization

Both Ridge and Lasso regression help improve model generalization by controlling the complexity, but they do so in different ways. Ridge is better when all features may contribute to the output, while Lasso is valuable for sparse models where only some features are relevant.

x_i	y_i	$x_i y_i$	x_i^2	$x_i^2 y$	x_i^3	x_i^4
1	1	1	1	1	1	1
2	4	8	4	16	8	16
3	9	27	9	81	27	81
4	15	60	16	240	64	256
$\sum x_i = 10$	$\sum y_i = 29$	$\sum x_i y_i = 96$	$\sum x_i^2 = 30$	$\sum x_i^2 y_i = 338$	$\sum x_i^3 = 100$	$\sum x_i^4 = 354$

$$\boldsymbol{a = X^{-1} B} \quad \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 4 & 10 & 30 \\ 10 & 30 & 100 \\ 30 & 100 & 354 \end{bmatrix}^{-1} \times \begin{bmatrix} 29 \\ 96 \\ 338 \end{bmatrix} = \begin{pmatrix} -0.75 \\ 0.95 \\ 0.75 \end{pmatrix}$$

$$y = -0.75 + 0.95\,x + 0.75\,x^2$$