Fundamentals of Machine Learning [DSE 2222]

Department of Data Science and Computer Applications
MIT, Manipal

January 2025

Slide Set 4 – Trees, Forests, Bagging, and Boosting

Classification and Regression Trees (CART)

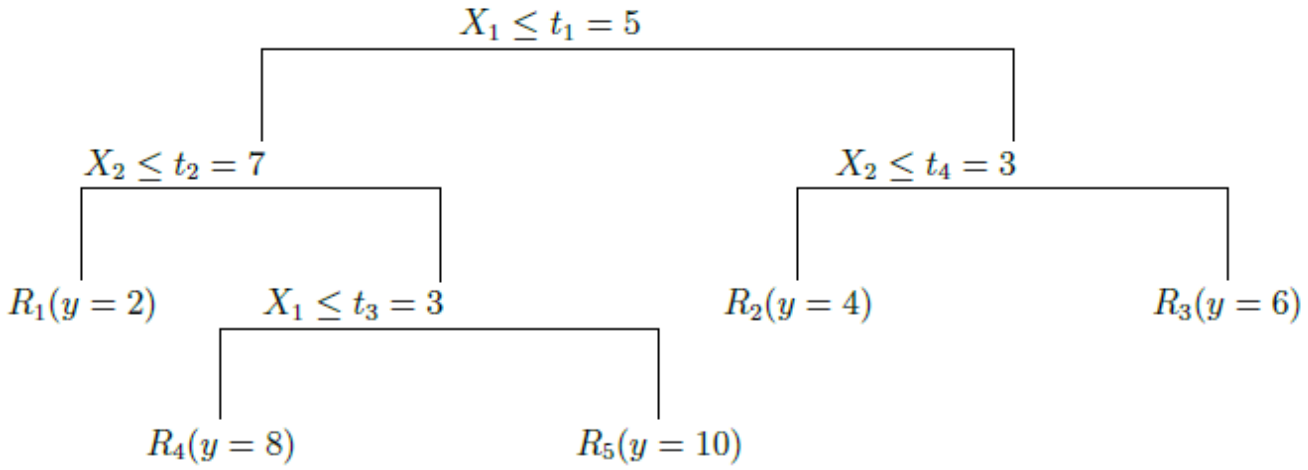
- Classification and regression trees or CART models, also called decision trees, are defined by recursively partitioning the input space, and defining a local model in each resulting region of input space.

Model definition:

- We start by considering regression trees, where all inputs are real-valued.
- The tree consists of a set of nested decision rules.
- At each node i, the feature dimension d_i of the input vector x is compared to a threshold value t_i , and the input is then passed down to the left or right branch, depending on whether it is above or below threshold.
- At the leaves of the tree, the model specifies the predicted output for any input that falls into that part of the input space.

CART

- The first node asks if x_1 is less than some threshold t_1. If yes, we then ask if x_2 is less than some other threshold t_2 . If yes, we enter the bottom left leaf node.
- This corresponds to the region of space defined by: $R_1 = \{\boldsymbol{x} : x_1 \leq t_1, x_2 \leq t_2\}$



CART

- CART is designed for two types of problems:
- **Classification Trees** - Used for categorical target variables (e.g., predicting "Yes" or "No").
- **Regression Trees** - Used for continuous target variables (e.g., predicting house prices).

The difference is:

- **Splitting criteria:** Classification uses **Gini impurity**, whereas Regression uses **Mean Squared Error (MSE)**.
- **Output values:** Classification trees give class labels; regression trees give continuous numerical values.

CART

- Formally, a regression tree can be defined by:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{j=1}^{J} w_j \mathbb{I}\left(\boldsymbol{x} \in R_j\right)$$

- where R_j is the region(partition of the input space) specified by the j^{th} leaf node, w_j is the predicted output for that node
- f(x;θ) is the predicted output for input x with parameters θ and θ = {(R_j , w_j) : j = 1 : J}, where J is the number of nodes.

- I(x∈R_j) is an indicator function, which is **1 if x belongs to region R_j, otherwise 0**.

CART

- Represents the calculation of w_j, which is the predicted value for a region R_j in a **Regression Tree**.

$$w_j = \frac{\sum_{n=1}^{N} y_n \mathbb{I}(\boldsymbol{x}_n \in R_j)}{\sum_{n=1}^{N} \mathbb{I}(\boldsymbol{x}_n \in R_j)}$$

- It is the average output (mean value) of all training samples that belong to region R_j.
- Sums up the actual target values y_n for all training samples that belong to region R_j
- Counts the number of training samples in R_j .

CART- Interpretation in Decision Trees

- The decision tree partitions the input space into J disjoint regions R_j
- Each region is associated with a constant output value w_j , which is either:
  - A class label (for classification).
  - A numeric value (mean target value in region R_j) (for regression).
- The indicator function ensures that the function $f(x;\theta)$ assigns the corresponding output w_j depending on which region x belongs to.

How this works in CART?

- The dataset is split into J non-overlapping regions R_j .
- Each region is assigned a constant prediction w_j , which is:
  - The majority class (for classification).
  - The mean target value (for regression).
- When a new data point x is given, the function determines which region R_j it falls into and assigns the corresponding value w_j .

CART – Example (Regression)

Consider a regression tree predicting house prices based on area size.

- The dataset is split into three regions:
  - R_1: $\text{Size} < 1000$ sqft → Predicts price $w_1 = 50$ Lakhs
  - R_2: $1000 \leq \text{Size} < 2000$ sqft → Predicts price $w_2 = 80$ Lakhs
  - R_3: $\text{Size} \geq 2000$ sqft → Predicts price $w_3 = 120$ Lakhs

For a house of **1500 sqft**, the function selects R_2, so:

$$f(1500; \theta) = w_2 = 80 \text{ Lakhs}$$

CART

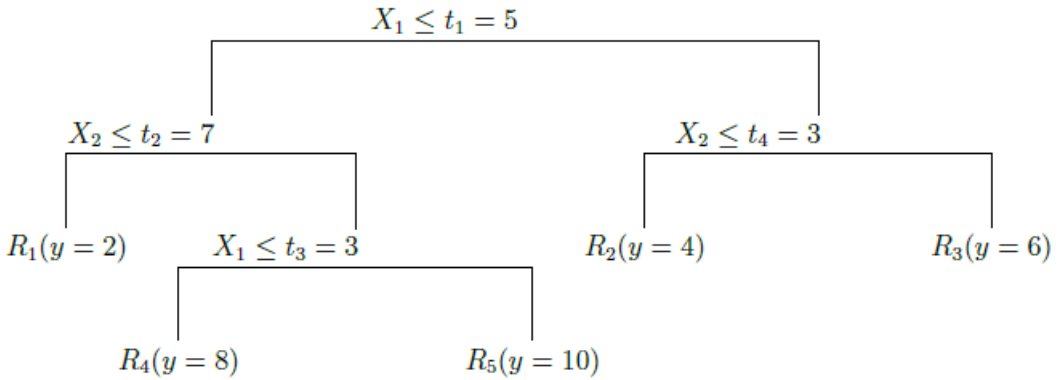
- The symbol $\mathrm{I}(x \in R_j)$ in the equation represents an **indicator function**. The indicator function is defined as:

$$I(x \in R_j) = \begin{cases} 1, & \text{if } x \in R_j \\ 0, & \text{otherwise} \end{cases}$$

- This function is used to determine whether a given input x belongs to the region R_j. If x is in R_j, the function returns 1; otherwise, it returns 0.

CART

- The regions themselves are defined by the feature dimensions that are used in each split, and the corresponding thresholds, on the path from the root to the leaf.
- For example, in Figure, we have
  - $R_1 = [(x_1 \leq t_1), (x_2 \leq t_2)]$,
  - $R_4 = [(x_1 \leq t_1), (x_2 > t_2), (x_3 \leq t_3)]$
- For classification problems, the leaves contain a distribution over the class labels, rather than just the mean response



CART

Model fitting:

To fit the model, we need to minimize the following loss:

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{n=1}^{N} \ell(y_n, f(\boldsymbol{x}_n; \boldsymbol{\theta})) = \sum_{j=1}^{J} \sum_{\boldsymbol{x}_n \in R_j} \ell(y_n, w_j)$$

- This is not differentiable
- The standard practice is to use a greedy procedure, in which we iteratively grow the tree one node at a time. (greedy, recursive, top-down) .
- This approach is used by CART

CART

- Suppose we are at node i; let $\boldsymbol{D_i} = \{(\boldsymbol{x_n}, \boldsymbol{y_n}) \in \boldsymbol{N_i}\}$ be the set of examples that reach this node.
- We will consider how to split this node into a left branch and right branch so as to minimize the error in each child subtree.
- If the $\boldsymbol{j^{th}}$ feature is a real-valued scalar, we can partition the data at node $\boldsymbol{i}$ by comparing to a threshold $\boldsymbol{t}$.
- *The set of possible thresholds T_j or feature j can be obtained by sorting the unique values of $\{x_{nj}\}$*
- For example, if feature 1 has the values {4.5,−12, 72,−12}, then we set T_1 = {−12, 4.5, 72}.
- For each possible threshold, we define the left and right splits

$$\mathcal{D}_i^L(j,t) = \{(\boldsymbol{x}_n, y_n) \in N_i : x_{n,j} \leq t\}$$

$$\mathcal{D}_i^R(j,t) = \{(\boldsymbol{x}_n, y_n) \in N_i : x_{n,j} > t\}$$

CART

- If the j^{th} feature is categorical, with K_j possible values, then we check if the feature is equal to each of those values or not.
- This defines a set of K_j possible binary splits:
  - $D_i^L\ (j,t) = \{(x_n, y_n) \in N_i : x_n, j = t\}$ and
  - $D_i^R\ (j,t) = \{(x_n, y_n) \in N_i : x_n, j \neq t\}$
- Once we have computed D^L i (j, t) and D^R i (j, t) for each j and t at node i, we choose the best feature j_i to split on, and the best value for that feature, t_i, as follows:

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^L(j,t)) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^R(j,t))$$

The goal is to minimize (j,t), meaning the split should create purer child nodes.

$$(j_i, t_i) = \arg \min_{j \in \{1,\ldots,D\}} \min_{t \in \mathcal{T}_j} \frac{|\mathcal{D}_i^L(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^L(j,t)) + \frac{|\mathcal{D}_i^R(j,t)|}{|\mathcal{D}_i|} c(\mathcal{D}_i^R(j,t))$$

- D_i is the set of all samples at node i.
- The goal is to minimize the weighted impurity (or cost function) after splitting.
- The first summation term represents the contribution of the left child node $D_i^L(j,t)$
- The second summation term represents the contribution of the right child node $D_i^R(j,t)$.
- The weight of each term is given by the proportion of samples in the left and right child nodes, respectively.
- $c(D)$ represents the impurity or cost function, which could be Gini impurity, entropy, or variance depending on the problem (classification or regression).

This equation finds the best feature j_i and its best threshold t_i by minimizing the total impurity after the split.

CART

- For regression, we can use the mean squared error:

$$\text{cost}(\mathcal{D}_i) = \frac{1}{|\mathcal{D}|} \sum_{n \in \mathcal{D}_i} (y_n - \overline{y})^2$$

where $\overline{y} = \frac{1}{|\mathcal{D}|} \sum_{n \in \mathcal{D}_i} y_n$ is the mean of the response variable for examples reaching node i

- For classification, we first compute the empirical distribution over class labels for this node:

$$\hat{\pi}_{ic} = \frac{1}{|\mathcal{D}_i|} \sum_{n \in \mathcal{D}_i} \mathbb{I}(y_n = c)$$

CART

- Given this, we can then compute the Gini index:

$$G_i = \sum_{c=1}^{C} \hat{\pi}_{ic}(1 - \hat{\pi}_{ic}) = \sum_{c} \hat{\pi}_{ic} - \sum_{c} \hat{\pi}_{ic}^2 = 1 - \sum_{c} \hat{\pi}_{ic}^2$$

- This is the expected error rate.
- $\hat{\pi}_{ic}$ is the probability a random entry in the leaf belongs to class c, and $1 - \hat{\pi}_{ic}$ is the probability it would be misclassified.
- Given one of the above cost functions, we can use Equation to pick the best feature, and best threshold at each node.
- We then partition the data, and call the fitting algorithm recursively on each subset of the data.

CART

- Alternatively we can define cost as the entropy or deviance of the node:

$$H_i = \mathbb{H}(\hat{\pi}_i) = -\sum_{c=1}^{C} \hat{\pi}_{ic} \log \hat{\pi}_{ic}$$

- A node that is pure (i.e., only has examples of one class) will have 0 entropy

- Common impurity measures include **Gini impurity** (for classification), **entropy** (for classification), and **variance reduction** (for regression n).

CART

Regularization

- If we let the tree become deep enough, it can achieve 0 error on the training set (assuming no label noise), by partitioning the input space into sufficiently small regions where the output is constant.
- However, this will typically result in overfitting. To prevent this, there are two main approaches.
- The first is to stop the tree growing process according to some heuristic, such as having too few examples at a node, or reaching a maximum depth.
- The second approach is to grow the tree to its maximum depth, where no more splits are possible, and then to prune it back, by merging split subtrees back into their parent. This can partially overcome the greedy nature of top-down tree growing.

CART

Handling missing input features

- The standard heuristic for handling missing inputs in decision trees is to look for a series of “backup” variables, which can induce a similar partition to the chosen variable at any given split;
- These can be used in case the chosen variable is unobserved at test time. These are called surrogate splits.
- This method finds highly correlated features, and can be thought of as learning a local joint model of the input.

CART

Pros and cons:

Tree models are popular for several reasons:

- They are easy to interpret.
- They can easily handle mixed discrete and continuous inputs.
- They are insensitive to monotone transformations of the inputs (because the split points (Ranking: **sorting the feature values** and finding the optimal threshold (for numerical attributes) or category grouping), so there is no need to standardize the data. They perform automatic variable selection.
- They are relatively robust to outliers.
- They are fast to fit and scale well to large data sets.
- They can handle missing input features.

CART

Disadvantages

- The primary one is that they do not predict very accurately compared to other kinds of model. This is in part due to the greedy nature of the tree construction algorithm.
- A related problem is that trees are unstable: small changes to the input data can have large effects on the structure of the tree, due to the hierarchical nature of the tree-growing process, causing errors at the top to affect the rest of the tree.

How CART Works

- **Splitting the Data**
  - The CART algorithm recursively partitions the dataset into two child nodes at each step. The best split is chosen using:
  - **Gini impurity (for classification)**
  - **Mean Squared Error (for regression)**
  - It continues splitting until a stopping condition is met (e.g., maximum depth, minimum samples per leaf).
- **Choosing the Best Split (Splitting Criteria)**
  - The CART algorithm evaluates every possible feature and threshold to find the best split.
  - **For Classification Trees: Gini Impurity**
  - Gini impurity measures how "pure" a node is:
  - $Gini = 1 - \sum p_i^2$ where p_i is the proportion of samples belonging to class i.
  - A lower Gini value means better purity.
  - The split that minimizes the weighted sum of Gini impurities in child nodes is chosen.

How CART Works

- **For Regression Trees: Mean Squared Error (MSE)**
- MSE is used to measure the variance within each node:

$$MSE = \frac{1}{n}\sum(y_i - \bar{y})^2$$

- Difference of the actual values and mean is considered.
- The split that minimizes MSE is selected.

CART – Solved Example 1

- A company wants to predict whether a customer will buy a product based on their **age** and **income**. The dataset is as follows:

Customer	Age	Income($1000)	Buy(Yes=1,No=0)
1	25	40	0
2	45	50	1
3	35	60	1
4	30	30	0
5	50	70	1

- The goal is to build a **decision tree** using CART to predict whether a new customer will buy the product or not.

CART – Solved Example 1

- The CART algorithm chooses splits based on **Gini Index** (for classification) or **Mean Squared Error (MSE)** (for regression).
- The Gini Index is calculated as: G= $1 - \sum_{c} \hat{\pi}_{ic}^2$
- **Consider splitting on "Age" at 35:**
  - Left Node: Customers with **Age ≤ 35** → { (25, 40, 0), (35, 60, 1), (30, 30, 0) }
  - Right Node: Customers with **Age > 35** → { (45, 50, 1), (50, 70, 1) }
- **Gini for Left Node:**
  - 2 customers don't buy (0) → p_0=2/3
  - 1 customer buys (1) → p_1=1/3
  - $Gini_{Left}$=1−(2/3)^2−(1/3)^2=1−4/9=0.44

Gini for Right Node:
•2 customers buy (1) → p1=2/2=1
$Gini_{Right}$=1−(1)^2=0

Weighted Gini: $GiniSplit = w_1 \cdot Gini_1 + w_2 \cdot Gini_2$

$Gini_{Split}$=3/5(0.44)+2/5(0)=0.264

If this is the best split (lowest Gini), then the first split happens at **Age ≤ 35**.

For each resulting node, repeat the process by considering the next best split (e.g., splitting on **Income**).

CART - Solved Example 2 (Classification Problem)

- Consider a dataset with two features (X1 and X2) and a binary target variable (Y). The goal is to build a decision tree using the CART algorithm to classify the data.

X1	X2	Y
1	2	0
2	3	0
3	4	1
4	5	1
5	6	0
6	7	1

CART - Solved Example 2

- The CART algorithm uses the **Gini Impurity** to decide the best split. The formula for Gini Impurity is:

$$1 - \sum_{c} \hat{\pi}_{ic}^2$$

Evaluate all possible splits for each feature (X1 and X2) and choose the one with the lowest Gini Impurity

Split on X1:For a given feature X, the split points are typically chosen at the midpoints between consecutive unique values in sorted order.

- **Split at X1 = 2.5**:
  - Left Node: X1 ≤ 2.5 → Y = [0, 0]
  - Right Node: X1 > 2.5 → Y = [1, 1, 0, 1]

- Gini for Left Node: $Gini_{left}=1-(2/2)^2-(0/2)^2=0$
- Gini for Right Node: $Gini_{right}=1-(3/4)^2-(1/4)^2=0.375$
- Weighted Gini: $Gini_{split}=(2/6)*0+(4/6)*0.375=$ **0.25**

CART - Solved Example 2

Split at X1 = 4.5:

- Left Node: X1 ≤ 4.5 → Y = [0, 0, 1, 1]
- Right Node: X1 > 4.5 → Y = [0, 1]
- Gini for Left Node: $Gini_{left}=1-(2/4)^2-(2/4)^2=0.5$
- Gini for Right Node:$Gini_{right}=1-(1/2)^2-(1/2)^2=0.5$
- Weighted Gini: $Gini_{split}=(4/6)\cdot 0.5+(2/6)\cdot 0.5=$ **0.5**

The best split for X1 is at **X1 = 2.5** with a Gini Impurity of **0.333**.

CART - Solved Example 2

Split on X2:

- **Split at X2 = 3.5**:
  - Left Node: X2 ≤ 3.5 → Y = [0, 0] Right Node: X2 > 3.5 → Y = [1, 1, 0, 1]
- Gini for Left Node: $Gini_{left}=1-(2/2)^2-(0/2)^2=0$
- Gini for Right Node:$Gini_{right}=1-(3/4)^2-(1/4)^2=0.25$
- Weighted Gini: $Gini_{split}=(2/6)\cdot 0+(4/6)\cdot 0.375=0.25$
- **Split at X2 = 5.5**:
  - Left Node: X2 ≤ 5.5 → Y = [0, 0, 1, 1] Right Node: X2 > 5.5 → Y = [0, 1]
- Gini for Left Node: $Gini_{left}=1-(2/4)^2-(2/4)^2=0.5$
- Gini for Right Node: $Gini_{righ}t=1-(1/2)^2-(1/2)^2=0.5$
- Weighted Gini: $Gini_{split}=(4/6)\cdot 0.5+(2/6)\cdot 0.5=0.5$

The best split for X2 is at **X2 = 3.5** with a Gini Impurity of **0.25**.

CART - Solved Example 2

Choose the Best Feature to Split

- Both X1 and X2 have the same Gini Impurity (0.25) for their best splits. We can choose either feature for the first split. Let's choose **X1 = 2.5**.

Build the Tree

1. **Root Node**: Split on X1 ≤ 2.5.
    1. Left Child: Pure node (Y = [0, 0]).
    2. Right Child: Split further.
2. **Right Child**: Split on X2 ≤ 3.5.
    1. Left Child: Pure node (Y = [1, 1]).
    2. Right Child: (Y = [0, 1]). Split further depending on the stopping condition

CART - Solved Example 2

- **Final Decision Tree:**
- If **X1 ≤ 2.5**, predict **Y = 0**.
- If **X1 > 2.5**:
  - If **X2 ≤ 3.5**, predict **Y = 1**.
  - If **X2 > 3.5**, predict **Y = 0** or **Y = 1** (depending on the majority class in the training data).

CART - Solved Example 3 (Regression Problem)

- In regression, the goal is to predict a continuous target variable instead of a class label.
- Consider a dataset with one feature (X) and a continuous target variable (Y). The goal is to build a regression tree to predict Y based on X.

X	Y
1	2
2	3
3	4
4	5
5	6
6	7

CART - Solved Example 3 (Regression Problem)

- The CART algorithm for regression minimizes the **variance** of the target variable in each split. The formula for variance is:

$$\text{Variance}(S) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \bar{y})^2$$

- We evaluate all possible splits for the feature (X) and choose the one that minimizes the weighted variance of the two resulting subsets.

Possible Splits:

- **Split at X = 1.5**:
  - Left Node: X ≤ 1.5 → Y = [2]
  - Right Node: X > 1.5 → Y = [3, 4, 5, 6, 7]
- Variance for Left Node: $\text{Variance}_{\text{left}} = 0$ (only one value)
- Variance for Right Node: $\bar{y}_{\text{right}} = (3+4+5+6+7)/5 = 5$

  $\text{Variance}_{\text{right}} = (3-5)^2+(4-5)^2+(5-5)^2+(6-5)^2+(7-5)^2/5 = 2$
- Weighted Variance: $\text{Variance}_{\text{split}} = (1/6)\cdot 0+(5/6)\cdot 2 =$ 1.667

CART - Solved Example 3 (Regression Problem)

Split at X = 2.5:

- Left Node: X ≤ 2.5 → Y = [2, 3]
- Right Node: X > 2.5 → Y = [4, 5, 6, 7]
- Variance for Left Node: $\bar{y}_{\text{left}}=(2+3)/2=2.5$

  $\text{Variance}_{\text{left}}=(2-2.5)^2+(3-2.5)^2/2=0.25$
- Variance for Right Node: $\bar{y}_{\text{right}}=(4+5+6+7)/4=5.5$

  $\text{Variance}_{\text{right}}=(4-5.5)^2+(5-5.5)^2+(6-5.5)^2+(7-5.5)^2/4=1.25$
- Weighted Variance: $\text{Variance}_{\text{split}}=(2/6)\cdot 0.25+(4/6)\cdot 1.25=$ 0.917

CART - Solved Example 3 (Regression Problem)

- **Split at X = 3.5**:
  - Left Node: X ≤ 3.5 → Y = [2, 3, 4]
  - Right Node: X > 3.5 → Y = [5, 6, 7]
- Variance for Left Node: $\bar{y}_{\text{left}}$ =(2+3+4)/3=3
  $\text{Variance}_{\text{left}}$=$(2-3)^2$+$(3-3)^2$+$(4-3)^2$/3=0.667
- Variance for Right Node: $\bar{y}_{\text{right}}$ =5+6+73=6
  $\text{Variance}_{\text{right}}$=$(5-6)^2$+$(6-6)^2$+$(7-6)^2$/3=0.667
- Weighted Variance:
- $\text{Variance}_{\text{split}}$=(3/6)·0.667+(3/6)·0.667= 0.667

Split at X = 4.5:

1. Left Node: X ≤ 4.5 → Y = [2, 3, 4, 5]
2. Right Node: X > 4.5 → Y = [6, 7]

Variance for Left Node:$\bar{y}_{\text{left}}$ =(2+3+4+5)/4=3.5
$\text{Variance}_{\text{left}}$=$(2-3.5)^2$+$(3-3.5)^2$+$(4-3.5)^2$+$(5-3.5)^2$/4=1.25

Variance for Right Node:$\bar{y}_{\text{right}}$ =6+72=6.5
$\text{Variance}_{\text{right}}$=$(6-6.5)^2$+$(7-6.5)^2$/2=0.25

Weighted Variance:
$\text{Variance}_{\text{split}}$=(4/6)·1.25+(2/6)·0.25= 0.917

CART - Solved Example 3 (Regression Problem)

- **Split at X = 5.5**:
  - Left Node: X ≤ 5.5 → Y = [2, 3, 4, 5, 6]
  - Right Node: X > 5.5 → Y = [7]
- Variance for Left Node: $\bar{y}_{\text{left}}$ =(2+3+4+5+6)/5=4
  $\text{Variance}_{\text{left}}=(2-4)^2+(3-4)^2+(4-4)^2+(5-4)^2+(6-4)^2/5=2$
- Variance for Right Node:
- $\text{Variance}_{\text{right}}$=0(only one value)
- Weighted Variance:
- $\text{Variance}_{\text{split}}=(5/6)\cdot 2+(1/6)\cdot 0=$ 1.667

The best split is the one with the **lowest weighted variance**. From the calculations above, the best split is at **X = 3.5** with a weighted variance of **0.667**.

X	Y
1	2
2	3
3	4
4	5
5	6
6	7

Final Step:
Build the Tree
Root Node: Split on X ≤ 3.5.

Left Child: Predict the mean of Y = [2, 3, 4], which is 3.
Right Child: Predict the mean of Y = [5, 6, 7], which is 6.

Final Regression Tree:
If X ≤ 3.5, predict Y = 3.
If X > 3.5, predict Y = 6.

ID3 - Iterative Dichotomiser 3

- The ID3 algorithm works by recursively partitioning the dataset based on attribute values until it reaches a pure subset or a stopping condition.
- It constructs a decision tree by choosing the best attribute at each step
- It follows a **top-down, divide-and-conquer approach** and selects attributes based on **information gain**
- "Iterative Dichotomiser" refers to the repeated process of dividing data into two distinct groups
- "iterative" means repeating the process, while "dichotomiser" means dividing into two parts, so the name describes how the algorithm works by repeatedly splitting data into smaller subsets based on specific attributes

ID3

Information Theory and Logarithm

In **Shannon's Information Theory**, the amount of information gained from an event occurring is given by:

$$I(x) = -\log_b P(x)$$

- I(x) is the information content (measured in bits if log base 2 is used),
- P(x) is the probability of event x,
- B is the base of the logarithm (commonly 2 for binary systems).

A **high-probability event** provides **less information**, while a **low-probability event** provides **more information**.

ID3

Entropy (Measure of Impurity)

- Entropy measures the **uncertainty** or **impurity** in a dataset. It is given by:

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

where:
- S is the dataset,
- p_i is the proportion of class i in S,
- n is the number of classes.
- The **log function in entropy** is used to measure **information content** or **uncertainty** in a system

ID3

Calculating Entropy

Consider a dataset with **10 instances**, classified into **two classes**:

- **6 instances** belong to Class **A**
- **4 instances** belong to Class **B**

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

$$= -\left(0.6 \times -0.736 + 0.4 \times -1.322\right)$$

$$= 0.971$$

ID3

Information gain

- The **information gain** measures how much entropy is reduced by splitting on a particular attribute.(after splitting the dataset on a specific feature.)

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

where:

- S_v is the subset where attribute A has value v
- $\frac{|Sv|}{|S|}$ is the proportion of instances in S with value v
- $H(S_v)$ is the entropy of the subset.

Higher Information Gain indicates that the feature splits the data more effectively, leading to more homogeneous subsets with respect to the target variable.

Suppose you are building a decision tree to predict whether students pass or fail based on the "Hours of Study" and "Participation."

- Compute the entropy of the dataset.
- For each feature, calculate the Information Gain.
- Select the feature with the highest Information Gain as the splitting criterion at each node.

ID3

Calculating Information Gain

- Let's say an attribute **"Weather"** has three values: **Sunny, Rainy, Overcast**. The dataset is split as follows:

Weather	Instances	Class A	Class B	Entropy
Sunny	5	3	2	0.971
Rainy	3	2	1	0.918
Overcast	2	2	0	0.000

$$\text{Gain}(S, \text{Weather}) = H(S) - \left(\frac{5}{10} \times 0.971 + \frac{3}{10} \times 0.918 + \frac{2}{10} \times 0.000 \right)$$

$$= 0.971 - (0.4855 + 0.2754 + 0)$$

$$= 0.210$$

If "Weather" has the **highest information gain**, it will be chosen as the root of the decision tree.

ID3 Solved Example

- Consider a dataset with two features (**Age** and **Income**) and a target variable (**BuysComputer**). The goal is to build a decision tree using the ID3 algorithm to classify whether a person buys a computer or not.

Age	Income	BuysComputer
Youth	High	No
Youth	Medium	No
Youth	Low	Yes
Middle	High	Yes
Middle	Low	Yes
Senior	Medium	No
Senior	Low	Yes

ID3 Solved Example

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

- For the target variable **BuysComputer**:
  - Total samples = 7
  - Yes = 4
  - No = 3
- Entropy of the dataset:
- $$\text{Entropy}(S) = -\left(\frac{4}{7}\log_2\left(\frac{4}{7}\right) + \frac{3}{7}\log_2\left(\frac{3}{7}\right)\right)$$

$$\text{Entropy}(S) = -\left(0.571 \cdot \log_2(0.571) + 0.428 \cdot \log_2(0.428)\right)$$

$$\text{Entropy}(S) = 0.985$$

ID3 Solved Example

- **Calculate Information Gain for Each Feature**

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

- **Feature: Age**
- Age has three values: Youth, Middle, and Senior.
- **Age = Youth**:
  - Samples = 3
  - Yes = 1
  - No = 2
  - Entropy(Youth)=−(1/3 $\log_2$(1/3)+2/3 $\log_2$(2/3))=0.918

ID3 Solved Example

- **Age = Middle**:
  - Samples = 2
  - Yes = 2
  - No = 0
  - Entropy(Middle)=0(pure node)
- **Age = Senior**:
  - Samples = 2
  - Yes = 1
  - No = 1
  - Entropy(Senior)=−(1/2 $\log_2$(1/2)+1/2 $\log_2$(1/2))=1

Weighted Entropy for Age:

- Weighted Entropy(Age)=(3/7·0.918)+(2/7·0)+(2/7·1)=0.536

Information Gain for Age:

- Information Gain(Age)=Entropy(S)−Weighted Entropy(Age)=0.985−0.536=0.449

ID3 Solved Example

Feature: Income

- Income has three values: High, Medium, and Low.
- **Income = High**:
  - Samples = 2
  - Yes = 1
  - No = 1
  - Entropy(High)=$-(1/2\log_2(1/2)+1/2\log_2(1/2))=1$

Income = Medium:

Samples = 2
Yes = 0
No = 2
Entropy(Medium)=0(pure node)

Income = Low:

Samples = 3
Yes = 3
No = 0
Entropy(Low)=0(pure node)

Weighted Entropy for Income:

Weighted Entropy(Income)=$(2/7\cdot1)+(2/7\cdot0)+(3/7\cdot0)=0.286$

Information Gain for Income:

Information Gain(Income)=Entropy(S)−Weighted Entropy(Income)=$0.985-0.286=0.699$

ID3 Solved Example

- The feature with the **highest Information Gain** is selected for the split. Here:
- Information Gain for **Age** = 0.449
- Information Gain for **Income** = 0.699
- The best feature to split on is **Income**.

Final Decision Tree:

- If **Income = Medium**, predict **No**.
- If **Income = Low**, predict **Yes**.
- If **Income = High**:
  - If **Age = Youth**, predict **No**.
  - If **Age = Middle**, predict **Yes**.
  - If **Age = Senior**, predict **No**.

ID3

How it works?

- **Calculate Entropy (Measure of Impurity)**
- **Calculate Information Gain**
- **Create Decision Node -** The attribute with the **highest information gain** is selected as the **splitting attribute**. A decision node is created, and branches are drawn for each attribute value.
- Repeat the Process Recursively
  - For each subset created by the split:
  - Compute entropy and information gain for the remaining attributes.
  - Choose the best attribute for the next split.
  - Continue splitting until one of the stopping conditions is met.
- The recursion stops when:
  - **Pure Node**: All instances in a subset belong to the same class.
  - **No More Attributes**: If all attributes are used but multiple classes remain, assign the **majority class**.
  - **Threshold on Depth/Instances**: To prevent **overfitting**, we can stop if the depth of the tree exceeds a predefined value.

ID3

Disadvantages

- **Overfitting** – ID3 can create very deep trees that fit training data perfectly but generalize poorly.
- **Cannot Handle Numerical Data Directly** – It works only with categorical data. C4.5 (an extension of ID3) can handle numerical data by converting it into categories.
- **Bias Toward Attributes with More Values** – Attributes with many unique values may have artificially high information gain

Improvements Over ID3

C4.5 Algorithm: Handles numerical data, missing values, pruning to prevent overfitting.

CART Algorithm: Constructs binary trees instead of multiway splits.

C4.5

- C4.5 is a popular algorithm used to generate decision trees, developed by **Ross Quinlan** as an extension of his earlier **ID3 (Iterative Dichotomiser 3)** algorithm.
- It is widely used for classification tasks in machine learning and data mining.
- It builds decision trees using the **Gain Ratio**, which addresses ID3's bias toward attributes with many values.
- Foundation for modern decision tree methods like **C5.0 and Random Forests**.

Characteristics

- Handles Continuous and Categorical Data: Unlike ID3, which only handles categorical attributes, C4.5 can manage continuous numerical attributes by creating threshold-based splits.

C4.5

- Deals with Missing Values: It assigns probabilities based on available data instead of discarding instances with missing values.
- Pruning to Avoid Overfitting: Uses pessimistic pruning, where branches that do not contribute significantly to accuracy are removed.
- Entropy and Information Gain Ratio:Unlike ID3, which uses information gain, C4.5 uses gain ratio to prevent bias toward attributes with many distinct values.
- Handles Noisy Data:By pruning and using statistical techniques, C4.5 can work well with imperfect datasets.
- Outputs Rules Instead of Trees:It can convert decision trees into a set of if-then rules for better interpretability.

C4.5

Entropy (Measure of Uncertainty)

- C4.5, like ID3, uses entropy to measure the impurity of a dataset. The entropy H(S) of a dataset S is given by:

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

- c is the number of classes.
- p_i is the proportion of instances belonging to class i.
- H(S) is **high** if classes are equally distributed and **low** if they are pure (i.e., one class dominates).

Information Gain (IG)

- For an attribute A, the **Information Gain** is calculated as:

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

- S_v is the subset of S where attribute A has value v.
- $|S_v|/|S|$ is the proportion of instances in S_v.
- $H(S_v)$ is the entropy of subset S_v.

- This measures how much uncertainty (entropy) is reduced after splitting on attribute A. **A higher Information Gain means a better attribute for splitting.**

C4.5

Split Information (SI)

- ID3 has a bias toward attributes with many distinct values.
- C4.5 overcomes this using **Split Information**, which measures how evenly the data is split:

$$SI(S, A) = -\sum_{v \subset A} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

- This acts as a normalization factor to prevent favoring attributes with too many unique values.

Gain Ratio (GR)

- To select the best attribute, C4.5 uses the **Gain Ratio**, which is defined as:

$$GR(S, A) = \frac{IG(S, A)}{SI(S, A)}$$

- **Higher Gain Ratio means a better attribute for splitting.**
- Unlike ID3, C4.5 prefers attributes that provide a balanced split, avoiding dominance by attributes with many unique values.

C4.5

Handling Continuous Attributes

- For continuous attributes, C4.5 selects a **threshold T** and splits data into **≤T and >T**.
  To find the best threshold, C4.5 evaluates:

$$H(S,T) = \frac{|S_{\leq T}|}{|S|} H(S_{\leq T}) + \frac{|S_{>T}|}{|S|} H(S_{>T})$$

- and selects the threshold T that maximizes Gain Ratio.
- Example: If Age = {22, 25, 28, 30, 35, 40}, C4.5 may find an optimal split at Age ≤ 28 and Age > 28.

Handling Missing Values

- If an attribute A has missing values, C4.5 uses **probability-weighted splitting**, where instances contribute to multiple branches proportionally based on observed distributions.

C4.5

Handling Missing Values

Example: If an attribute Wind has missing values, but in known cases:

- 80% of the samples have “Strong Wind”
- 20% have “Weak Wind”
- Then, missing-value instances are assigned to both branches with weights (0.8, 0.2).

Pruning (To Prevent Overfitting)

- C4.5 applies **pessimistic pruning**, which removes branches that don’t improve classification accuracy. It estimates error rates using:

$$e = \frac{E + 0.5}{N}$$

- E is the number of misclassified instances and N is the total instances at that node.
- If pruning improves estimated accuracy, the subtree is replaced with a leaf.

C4.5

Summary

- Compute **Entropy** H(S).
- Compute **Information Gain** IG(S,A)
- Compute **Split Information** SI(S,A)
- Compute **Gain Ratio** GR(S,A)
- Select the attribute with the highest **Gain Ratio**.
- Handle **continuous values** by finding an optimal threshold.
- Handle **missing values** using probability-weighted splits.
- Perform **pessimistic pruning** to remove overfitting branches.

C4.5 - Solved Numericals

- Consider a small dataset with **4 instances** for a **binary classification problem** (PlayTennis: Yes/No).

Outlook	Humidity	Wind	PlayTennis
Sunny	High	Weak	No
Sunny	Normal	Strong	Yes
Overcast	High	Weak	Yes
Rain	Normal	Weak	Yes

Solution:

- The **Entropy (H)** for PlayTennis: **3 Yes**, **1 No**
- Probability of "Yes" =p_{yes}=3/4=0.75
- Probability of "No" = p_{no}=1/4=0.25

$H(S)=-(0.75\log_2 0.75+0.25\log_2 0.25)$ $H(S) = -\sum p_i \log_2 p_i$

$H(S)=-(0.75\times-0.415)-(0.25\times-2)$

$H(S)=0.311+0.5=0.811$

C4.5 - Solved Numericals

- Compute Information Gain for Attributes "Outlook" , "Humidity“, “Wind”.

Outlook:

- Overcast: 1 instance, all "Yes"
- Sunny: 2 instances, 1 "Yes", 1 "No"
- Rainy: 1 instance, all "Yes"
- Entropy for Outlook:

$H(outlook)=1/4\times0+2/4\times1+1/4\times0=0.5$

Information Gain:

$IG(Outlook)=H(S)-H(Outlook)$

$=0.811-0.5=0.311$

Humidity:

High: 2 instances, 1 "Yes", 1 "No"

Normal: 2 instances, all "Yes“

Entropy for Humidity:

$H(Humidity)=2/4\times1+2/4\times0=0.5$

Information Gain:

$IG(Humidity)=H(S)-H(Humidity)$

$=0.811-0.5=0.311$

Wind:

Weak: 3 instance, 2 "Yes“, 1 “No”

Strong: 1 instances, 1 "Yes“

$H(Wind)=3/4\times0.918+1/4\times0=0.688$

$IG(S,Wind)=0.811-0.688=0.123$

C4.5 - Solved Numericals

Compute Split Information: $$SI(S, A) = -\sum_{v \subset A} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

For **Outlook**:

SI(S,Outlook)=−(2/4$\log_2$ 2/4+1/4$\log_2$ 1/4+1/4$\log_2$1/4)

=−(0.5×−1+0.25×−2+0.25×−2)

=−(−0.5−0.5−0.5)=1.5

Compute Gain Ratio

GR(S,Outlook)=IG(S,Outlook)/SI(S,Outlook)

GR(S,Outlook)=0.311/1.5=0.207

GR(S,Humidity)=0.311/1.0=0.311

GR(S,Wind)=0.123/0.918=0.134

Select the Best Attribute

Since **Humidity** has the highest **Gain Ratio (0.311)**, we split on **Humidity**

first.

- If **Humidity = Normal**, all samples are "Yes" → **Leaf Node: Yes**
- If **Humidity = High**, we continue splitting using **Outlook or Wind**

If all attributes have same information gain, one can choose either attribute for the split or use additional criteria like the number of splits or domain knowledge.

Bootstrapping

- Bootstrapping is a **resampling technique(estimate the sampling distribution)** used to estimate properties (e.g., mean, variance, confidence intervals) of a population by drawing multiple samples with replacement from an observed dataset.
- Bootstrapping is a **sampling technique** where multiple datasets are created from the original dataset by randomly selecting samples **with replacement**. This means that some samples may appear multiple times, while others may not appear at all.
- If the original dataset has **N samples**, a bootstrap sample also has **N samples**, but some data points are repeated.
- It helps in estimating population statistics and reducing variance in models.
- Used in statistical resampling methods like **confidence interval estimation** and **hypothesis testing**.

Bootstrapping

- **The Bootstrapping Process Mathematically**
- Let's assume we have a dataset **D** with **N** observations:

$$D = \{X_1, X_2, X_3, \ldots\ldots, X_n\}$$

where each X_i is an independent and identically distributed random variable (observations in the dataset).

- A bootstrap sample **D*** is obtained by randomly drawing **N** values from **D with replacement**.

Formally, let: $D^* = X_1^*, X_2^*, \ldots\ldots XN^*\}$

where each X_i^* is randomly selected from D. Some data points in D may appear multiple times, while others may be omitted.

Bootstrapping

- To generate a **bootstrap sample**, we randomly select n data points from D **with replacement**. This means that some of the points may appear more than once in the bootstrap sample, while others may not appear at all.
- Formally, let the bootstrap sample be denoted by $D^* = X_1^*, X_2^*, \ldots \ldots XN^*\}$
- , where each X_i^* is chosen randomly from D. This means for each X_i^*, the probability of selecting any specific X_j from D is equal for all points.
- The probability of each X_j being selected for X_i^* is 1/n, and since we're selecting with replacement, the same element can appear multiple times in the bootstrap sample.

Estimation Process

- Now, we can use this bootstrap sample D* to estimate some statistic, such as the sample mean or sample variance. For example, the **mean** of the bootstrap sample D* is:

$$\hat{\mu}^* = \frac{1}{n}\sum_{i=1}^{n} X_i^*$$

- where $\hat{\boldsymbol{\mu}}^*$ is the bootstrap mean

Bootstrapping

- **Repeating the Process (B Replications)**
- To get an accurate estimate of the sampling distribution, we generate B different bootstrap samples, each with size n, and compute the statistic of interest for each sample.
- Each bootstrap sample is independent of others, and the process involves resampling with replacement.
- Let the b-th bootstrap sample's statistic be denoted as $\hat{\mu}^*_b$, where b=1,2,...,B.
- Then, the **distribution of the bootstrap statistics** can be approximated as the empirical distribution of the B estimates.

Bootstrapping

- **Bootstrap Distribution and Confidence Intervals**
- The collection of bootstrap estimates $\hat{\mu}^*_1$, $\hat{\mu}^*_2$,..... $\hat{\mu}^*_B$ approximates the **sampling distribution** of the statistic.
- From this distribution, we can compute quantities like the **mean** or **variance** of the bootstrap estimates, which are used to estimate the **bias** or **standard error** of the statistic.
- **Bootstrap Mean**: The mean of the bootstrap estimates:

$$\hat{\mu}_{\text{bootstrap}} = \frac{1}{B} \sum_{b=1}^{B} \hat{\mu}_b^*$$

- **Bootstrap Standard Error (SE)**: The standard deviation of the bootstrap estimates:

$$SE_{\hat{\mu}} = \sqrt{\frac{1}{B-1} \sum_{b=1}^{B} (\hat{\mu}_b^* - \hat{\mu}_{\text{bootstrap}})^2}$$

Bootstrapping

- **Confidence Intervals**: From the bootstrap distribution, we can compute the **percentile-based** confidence interval.
- For example, the 95% confidence interval for $\hat{\mu}$ could be the 2.5th and 97.5th percentiles of the boots trap estimates:

$$\text{CI} = \left[\hat{\mu}_{(2.5)}, \hat{\mu}_{(97.5)}\right]$$

- Where $\hat{\mu}_{(2.5)}$ and $\hat{\mu}_{(97.5)}$ represent the 2.5th and 97.5th percentiles of the sorted bootstrap statistics.

Bootstrapping

Mathematical Steps in Bootstrapping

- Start with the original dataset D of size n.
- Generate B bootstrap samples D^* by randomly selecting n data points from D with replacement.
- For each bootstrap sample, calculate the statistic of interest $\hat{\mu}_b^*$ (e.g., the mean, variance).
- Estimate the sampling distribution of the statistic by examining the set of B statistics $\hat{\mu}^*_1, \hat{\mu}^*_2, \ldots\ldots \hat{\mu}^*_B$
- Compute quantities like the bootstrap mean, standard error, and confidence intervals.

Bootstrapping is a powerful non-parametric method for estimating the distribution of a statistic, particularly when the true sampling distribution is unknown or difficult to compute analytically

Bagging(Bootstrap Aggregating)

- Bagging (Bootstrap Aggregating) is an ensemble learning technique in machine learning.
- It uses bootstrapping to create multiple training datasets, trains a separate model on each, and then combines their predictions.
- It improves stability by training multiple models on different bootstrap samples and aggregating their predictions.
- The goal of bagging is to reduce variance and improve stability in machine learning models.

Example of Bagging:

- In **Random Forests**, multiple **decision trees** are trained on different **bootstrap samples**, and their predictions are aggregated (majority voting for classification, averaging for regression).

Bagging(Bootstrap Aggregating)

Working Procedure

- Bootstrap Sampling:
  - Given a dataset of size n, multiple new datasets (bootstrap samples) are created by sampling with replacement.
  - Each bootstrap sample has the same size nn, but some data points may appear multiple times while others may be missing.
- Model Training:
  - A separate weak learner (typically the same type of model) is trained on each bootstrap sample.
  - These models learn slightly different patterns from the dataset.
- Aggregation of Predictions:
  - For classification: The final prediction is obtained by majority voting across all models.
  - For regression: The final prediction is obtained by averaging the predictions of all models.

Bagging(Bootstrap Aggregating)

Mathematical Formulation of Bagging

- Let: $D=\{(X_1,Y_1),(X_2,Y_2),\ldots,(X_n,Y_n)\}$ be the training dataset.
- We generate B bootstrap samples: $D_1^*,D_2^*,\ldots,D_B^*$
- Each weak learner $f_b(x)$ is trained on a bootstrap sample D_b^*.

The final prediction is given by:

For Regression:

$$\hat{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

(Averaging the predictions)

For Classification:

$$\hat{y} = \text{mode}\{f_1(x), f_2(x), \ldots, f_B(x)\}$$

(Taking the majority vote)

Difference between Bootstrapping and Bagging

Bootstrapping	Bagging
A statistical method for resampling with replacement	A machine learning ensemble method that uses bootstrapping
Estimates confidence intervals, bias, variance, etc.	Reduces variance and improves model performance
Used for statistics, hypothesis testing, parameter estimation	Used for machine learning (Random Forest, Bagging Classifier, etc.)
Creates multiple datasets from a single dataset	Trains multiple models on different bootstrapped datasets
No aggregation step	Uses aggregation (voting or averaging)

Bootstrapping alone does not perform aggregation, while bagging **combines multiple models' outputs** to improve predictions.

Ensemble Learning

- Ensemble learning is a machine learning paradigm where multiple models, often referred to as "weak learners," are combined to solve the same problem in order to achieve better performance than any single model could on its own.
- Ensemble learning combines the predictions of multiple models, the strengths of individual models can compensate for the weaknesses of others, resulting in improved accuracy and robustness.

Types of Ensemble Methods

- Bagging (Bootstrap Aggregating)
  - Reduces variance and prevents overfitting
  - Example: Random Forest, Bagged Decision Trees
- Boosting
  - Models are trained sequentially, where each model focuses on the errors made by the previous one. Weights are assigned to misclassified instances to improve future models.
  - Reduces bias and improves predictive accuracy.
  - Examples: AdaBoost, Gradient Boosting Machines (GBM), XGBoost, LightGBM,

Ensemble Learning

- Stacking (Stacked Generalization)
  - Combines multiple models by training a meta-model that learns how to best combine the predictions of base models
  - Any combination of base learners (e.g., decision trees, SVMs, neural networks) with a meta-model like logistic regression or another complex algorithm
- Voting
  - Combines the predictions of multiple models either through majority voting (for classification) or averaging (for regression).
  - **Types**:
    - **Hard Voting**: The class with the most votes is selected.
    - **Soft Voting**: Weighted averaging of probabilities or confidences is used.
  - Using logistic regression, SVM, and decision tree together for prediction.

Ensemble Learning

Disadvantages

- Ensembles can be computationally expensive to train and deploy.
- The final model is harder to interpret compared to individual models.
- If not designed properly, ensembles can still overfit the data.

Ensemble learning

- Decision trees can be quite unstable, in the sense that their predictions might vary a lot if the training data is perturbed. In other words, decision trees are a high variance estimator.
- A simple way to reduce variance is to average multiple models. This is called ensemble learning

$$f(y|\boldsymbol{x}) = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} f_m(y|\boldsymbol{x})$$

- where f_m is the m'th base model.
- The ensemble will have similar bias to the base models, but lower variance, generally resulting in improved overall performance
- Averaging is a sensible way to combine predictions from regression models. For classifiers, it can sometimes be better to take a majority vote on the outputs. (This is sometimes called a committee method.)

Ensemble Methods

- Suppose each base model is a binary classifier with an accuracy of θ, and suppose class 1 is the correct class.
- Let $Y_m \in \{0, 1\}$ be the prediction for the m'th model, $S = \sum_{m=1}^{M}$ be the number of votes for class 1.
- We define the final predictor to be the majority vote, i.e., class 1 if S > M/2 and class 0 otherwise.
- The probability that the ensemble will pick class 1 is

$$p = \Pr(S > M/2) = 1 - B(M/2, M, \theta)$$

where B(x,M, θ) is the cdf of the binomial distribution with parameters M and θ evaluated at x.

- For θ = 0.51 and M = 1000, we get p = 0.73 and with M = 10, 000 we get p = 0.97.
- The performance of the voting approach is dramatically improved, because we assumed each predictor made independent errors.
- In practice, their mistakes may be correlated, but as long as we ensemble sufficiently diverse models, we can still come out ahead.

Random Forest

- It uses **bagging** (Bootstrap Aggregating) and introduces randomness in feature selection, making it both robust and versatile.

Training Phase

- **Bootstrap Sampling**
  - Assume we have a dataset $D=\{(x_1,y_1),(x_2,y_2),\ldots,(x_n,y_n)\}$ of n samples.
  - For each tree T_k in the forest (k ranges from 1 to N_t, the total number of trees), create a bootstrap sample D_k by sampling n examples with replacement from D.
  - Some samples may appear multiple times in D_k, while others may be excluded.
- **Building the Decision Tree**
  - At each node of the tree:
    - Instead of considering all p features for splitting, randomly select a subset of mmm features (m<p).
    - Choose the best split from this subset of features using a splitting criterion like **Gini Impurity** (for classification) or **Mean Squared Error** (for regression).

Random Forest

- Grow the tree until a stopping condition is met:
  - Maximum depth.
  - Minimum number of samples per leaf.
  - No further reduction in impurity.

- **Repeat**
  - Repeat steps 1 and 2 for N_t trees, building an ensemble of N_t decision trees

Prediction Phase

- **For Classification:**
  - Each tree T_k outputs a class prediction y_k.
  - Final prediction y is made using majority voting: $y=\text{mode}(y_1,y_2,\ldots,y_{Nt})$
- **For Regression:**
  - Each tree T_k outputs a numer
  - Final prediction $\hat{y}$ is the aver

$$\hat{y} = \frac{1}{N_t} \sum_{k=1}^{N_t} \hat{y}_k$$

Random Forest

Gini Impurity (For Classification)

- The Gini Impurity is used to measure the quality of a split at a node. For a node containing n samples with C classes, the Gini Impurity is:

$$G = 1 - \sum_{i=1}^{C} p_i^2$$

- where p_i is the proportion of samples in class i at the node.
- The split that minimizes the weighted average Gini Impurity of the child nodes is chosen:

$$G_{\text{split}} = \frac{n_L}{n} G_L + \frac{n_R}{n} G_R$$

where:

- G_L and G_R are the Gini impurities of the left and right child nodes.
- n_L and n_R are the number of samples in the left and right child nodes.
- n is the total number of samples at the parent node.

Random Forest

Mean Squared Error (For Regression)

- For regression tasks, the splitting criterion minimizes the variance of the target values in the child nodes:

$$\text{MSE}_{\text{split}} = \frac{n_L}{n}\text{Var}(y_L) + \frac{n_R}{n}\text{Var}(y_R)$$

where:

- Var(y) is the variance of the target variable y in a node.

Random Forest

Out-of-Bag (OOB) Error

- Since each tree is trained on a bootstrap sample, approximately 1/3 of the samples are left out (not included in the bootstrap). These samples are called **out-of-bag samples**.
- For each data point, predict its label using only the trees that did not see it during training.
- Calculate the OOB error as:

$$\text{OOB Error} = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}(\hat{y}_i^{\text{OOB}} \neq y_i)$$

- where $\mathbb{I}$ is an indicator function, $\hat{y}_i^{OOB}$ is the OOB prediction, and y_i is the true label.

Random Forest

Advantages

- Handles Non-linearity: Trees naturally split the data based on non-linear boundaries
- Ensemble averaging reduces the risk of overfitting compared to individual trees.
- Random Forest can compute feature importance, giving insight into the most relevant features.

Random Forest algorithm for classification

Assume we have the following toy dataset with 6 data points and two features:

Data Point	X_1	X_2	Class(Y)
1	2	3	A
2	1	2	A
3	4	4	B
4	5	6	B

Random Forest algorithm for classification

Solution

Create Bootstrap Samples

- Assume we build **3 trees** (N_t=3) in the Random Forest

- **Bootstrap Sampling:**
- For **Tree 1**, we randomly sample the dataset with replacement: D_1={(1,A),(4,B),(3,B),(4,B),(5,A),(6,B)}
- For **Tree 2**, another bootstrap sample: D_2={(3,B),(6,B),(2,A),(3,B),(1,A),(5,A)}
- For **Tree 3**, another bootstrap sample: D_3={(4,B),(5,A),(1,A),(6,B),(5,A),(2,A)}

Random Forest algorithm for classification

Train Decision Trees

- Each tree is trained on its bootstrap sample, and only a **random subset of features** is considered at each split.

1. Tree 1:
At the root node, randomly select m=1 feature (e.g., X_1).

Compute **Gini Impurity** for splits based on X_1.
(Gini calculation given in next slide)
Split the data at X_1=2.5:
Left child: {(1,A),(5,A),(6,B)}
Right child: {(4,B),(3,B),(4,B)}
Repeat for child nodes until a stopping criterion is met.

2. Tree 2:
•At the root node, randomly select m=1 feature (e.g., X_2).

•Compute **Gini Impurity** for splits based on X_1.

•Split the data at X_2=3.5:
- Left child: {(1,A),(2,A),(5,A)}
- Right child: {(6,B),(3,B),(3,B)}

•Continue growing the tree.

3. Tree 3:
•Repeat the process for another bootstrap sample and random feature selection

Random Forest algorithm for classification

Numerical Gini Calculation (Example for Tree 1 Root Split)

$X_1=2.5$

- Suppose we split D_1 based on $X_1=2.5$ into:
- Left child: {(1,A),(5,A),(6,B)} Right child: {(4,B),(3,B),(4,B)}

Gini for Left Child:

Classes: {A,B}

$p_A=2/3$, $p_B=1/3$

$G_{Left}=1-(p_A^2+p_B^2)=1-((2/3)^2+(1/3)^2)$

$=1-(4/9+1/9)=4/9$

Gini for Right Child:

Classes: {B}

$p_B=1$

$GRight=1-(p_B^2)=1-(1^2)=0$

Weighted Gini for Split:

$G_{split}=3/6 \cdot G_{Left}+3/6 \cdot G_{right}$

$=3/6 \cdot 4/9+3/6 \cdot 0 =2/9$

This split reduces impurity, so it is chosen

Random Forest algorithm for classification

Prediction

- Prediction for a **new data point**: (X_1=2,X_2=4).
- **Tree 1**:
  - Traverses the decision tree and predicts A.
- **Tree 2**:
  - Traverses the decision tree and predicts B.
- **Tree 3**:
  - Traverses the decision tree and predicts A.

Final Prediction (Majority Voting):

- Predictions: A,B,A
- Majority class = A
- Thus, the Random Forest predicts **Class A** for the new data point.

Random Forest- Solved Numericals

Problem 1:Out-of-Bag (OOB) Error Calculation

A Random Forest model is trained with 100 trees. Each tree is trained on a bootstrap sample of the dataset. For a particular data point, 70 trees do not include it in their bootstrap sample. Out of these 70 trees, 50 predict class "A" and 20 predict class "B". Calculate the Out-of-Bag (OOB) error for this data point.

Solution:

- The OOB prediction for the data point is the majority vote of the trees that did not include it in their bootstrap sample.
- Majority prediction: Class "A" (since 50 > 20).
- If the true label for this data point is "B", the OOB error is 1 (incorrect prediction).
- If the true label is "A", the OOB error is 0 (correct prediction).

Random Forest- Solved Numericals

Problem 2: Feature Importance

A Random Forest model is trained on a dataset with 3 features: F_1,F_2,F_3. The Gini importance scores for the features are:

F_1:0.15, F_2:0.45, F_3:0.40 (If not given, to be calculated from the dataset)

Calculate the normalized feature importance scores.

Solution:

- Total importance = 0.15+0.45+0.40=1
- Normalized importance:
  - F_1=0.15/1.00=0.15
  - F_2=0.45/1.00=0.45
  - F_3=0.40/1.00=0.40

Random Forest- Solved Numericals

Problem 3: Probability Estimation

A Random Forest model consists of 200 trees. For a given data point, 120 trees predict class "A" and 80 trees predict class "B". What is the predicted probability for class "A"?

Solution:

- Probability of class "A" = Number of trees predicting "A" / Total number of trees.
- P(A)=120/200=0.6 or 60%.

Random Forest- Solved Numericals

Problem 4: Classification with Random Forest

A dataset has 1000 samples and 10 features. A Random Forest model is trained with 500 trees, and each tree is limited to using a maximum of 3 features. What is the total number of features considered across all trees?

Solution:

- Each tree uses 3 features.
- Total features considered = Number of trees × Features per tree.
- Total features=500×3=1500.

Random Forest- Solved Numericals

Problem 5: Entropy and Information Gain

A Random Forest model is trained on a dataset with 2 classes. At a particular node in one of the trees, the class distribution is:

- Class 0: 30 samples
- Class 1: 10 samples

Calculate the entropy at this node and the information gain if the node is split into two child nodes with the following class distributions:

- Child Node 1: Class 0 = 20, Class 1 = 5
- Child Node 2: Class 0 = 10, Class 1 = 5

Random Forest- Solved Numericals

Entropy at the parent node:

- Total samples = 30+10=40
- P(0)=30/40=0.75, P(1)=10/40=0.25
- Entropy = $-P(0)\log_2(P(0))-P(1)\log_2(P(1))$
- Entropy=$-0.75\log_2(0.75)-0.25\log_2(0.25)\approx 0.811$

Entropy at child nodes:

- Child Node 1: P(0)=20/25=0.8, P(1)=5/25=0.2
  - Entropy = $-0.8\log_2(0.8)-0.2\log_2(0.2)\approx 0.722$
- Child Node 2: P(0)=10/15=0.666, P(1)=5/15=0.333
  - Entropy = $-0.666\log_2(0.666)-0.333\log_2(0.333)\approx 0.918$

Weighted entropy of child nodes:

Weighted entropy = $25/40\times 0.722+15/40\times 0.918\approx 0.798$

Information gain:

Information gain = Entropy(parent) - Weighted entropy(child nodes).

Information gain=$0.811-0.798=0.013$

Random Forest- Solved Numericals

Problem 6: Random Forest Hyperparameters

A Random Forest model is trained with the following hyperparameters:

- Number of trees (n_estimators) = 100
- Maximum depth of each tree (max_depth) = 10
- Minimum samples required to split a node (min_samples_split) = 5
- Minimum samples required at a leaf node (min_samples_leaf) = 2

If the dataset has 10,000 samples, what is the maximum number of splits that can occur in the entire forest?

Solution:

- Each tree can have a maximum depth of 10, so the maximum number of splits per tree is $2^{10}-1=1023$
- Total splits in the forest = Number of trees × Splits per tree.

$$\text{Total splits}=100\times1023=102,300.$$

END