Ensemble Learning

Sachin Tripathi

IIT(ISM), Dhanbad

Topics to be covered

- ❑ Introduction to Ensemble Methods and Types
- ❑ Random Forests
- ❑ Gradient Boosting

Definition

Ensemble learning helps improve machine learning results by combining several machine learning models

The two main classes of ensemble learning methods are

- Bagging
- Boosting

Why Ensemble ??

There are two main reasons to use an ensemble over a single model, and they are related; they are:–

Performance: An ensemble can make better predictions and achieve better performance than any single contributing model.
Robustness: An ensemble reduces the spread or dispersion of the predictions and model performance

Model Error

The error emerging from any machine model can be broken down into three components mathematically.

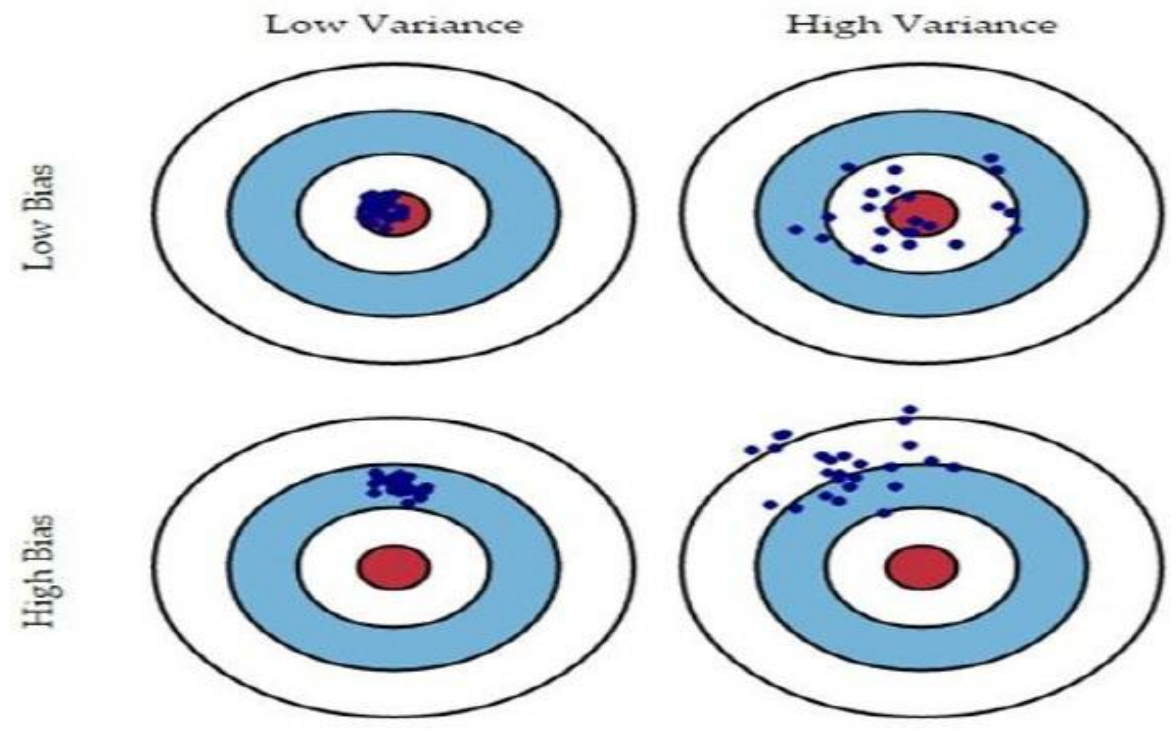
$$Err(x) = \left(E[\hat{f}(x)] - f(x)\right)^2 + E\left[\hat{f}(x) - E[\hat{f}(x)]\right]^2 + \sigma_e^2$$

$$Err(x) = \mathrm{Bias}^2 + \mathrm{Variance} + \mathrm{Irreducible\ Error}$$

• Bias error is useful to quantify how much on an average are the predicted values different from the actual value. A high bias error means we have an under-performing model which keeps on missing essential trends.

• Variance on the other side quantifies how are the prediction made on the same observation different from each other. A high variance model will over-fit on your training population and perform poorly on any observation beyond training

Bias and Variance



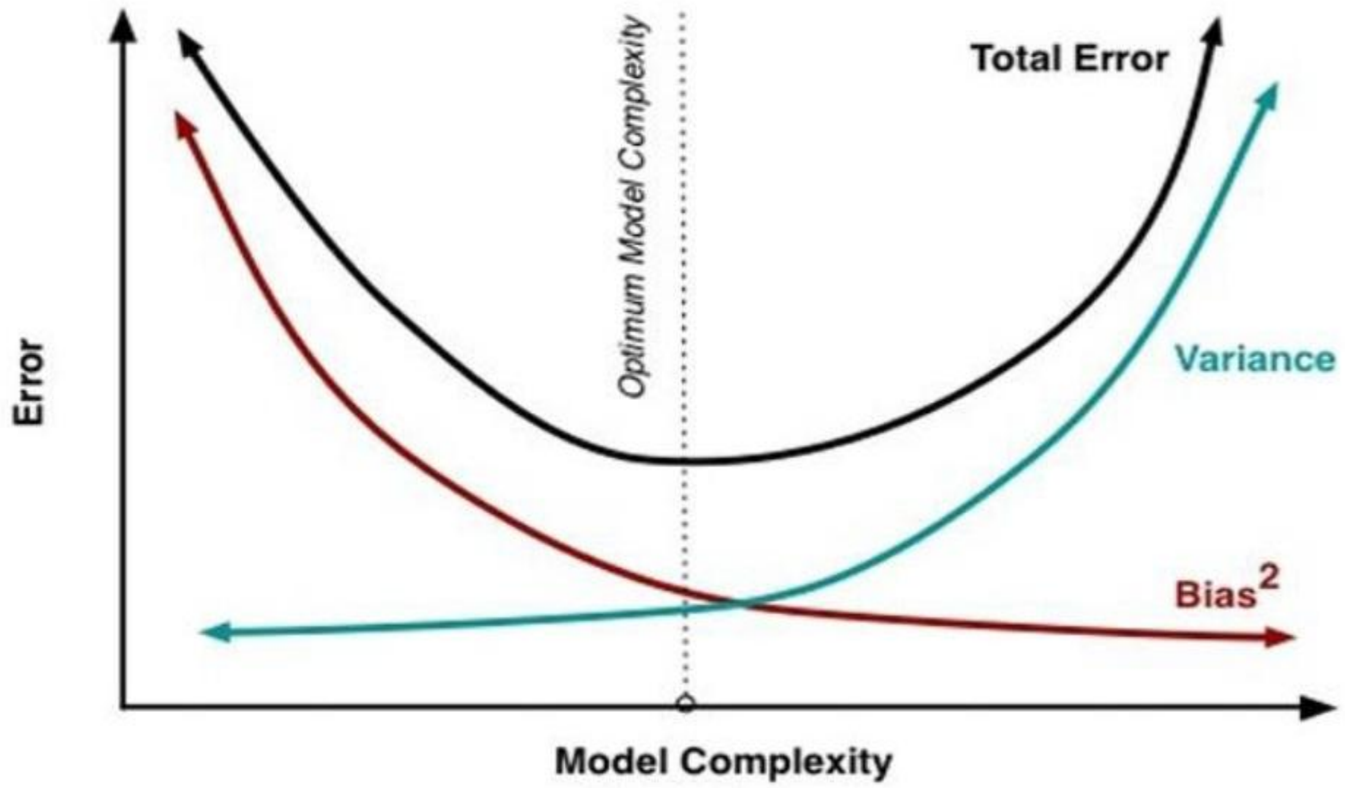
Bias –Variance Trade-off

Normally, as you increase the complexity of your model, you will see a reduction in error due to lower bias in the model. However, this only happens till a particular point.

• As you continue to make your model more complex, you end up over-fitting your model and hence your model will start suffering from high variance.

• A champion model should maintain a balance between these two types of errors. This is known as the trade-off management of bias-variance errors. Ensemble learning is one way to execute this trade off analysis

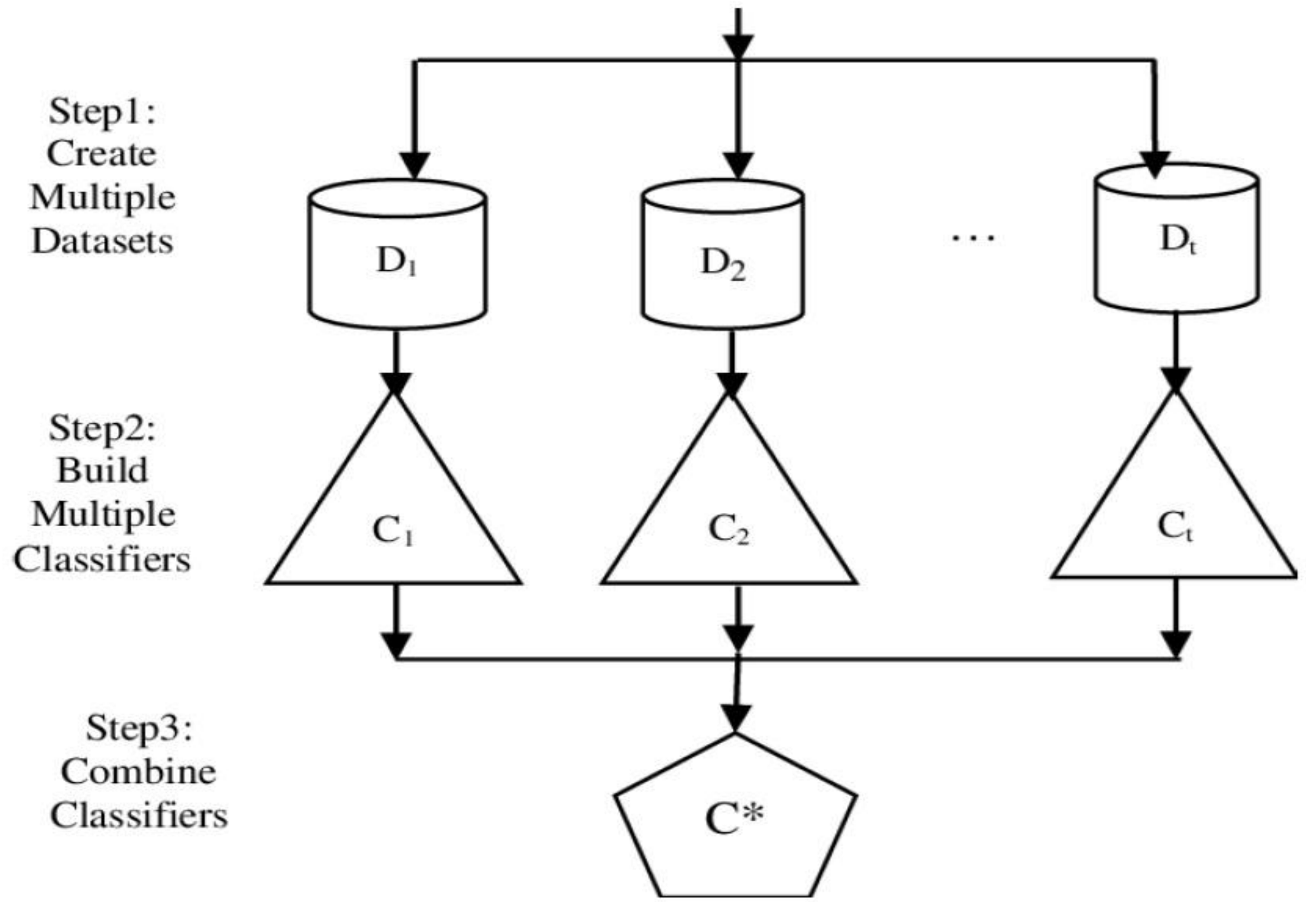
Bias Variance Trade-off



Ensemble Creation Approaches

- Unweighted Voting (e.g. Bagging)
- Weighted voting – based on accuracy (e.g. Boosting), Expertise, etc.
- Stacking - Learn the combination function

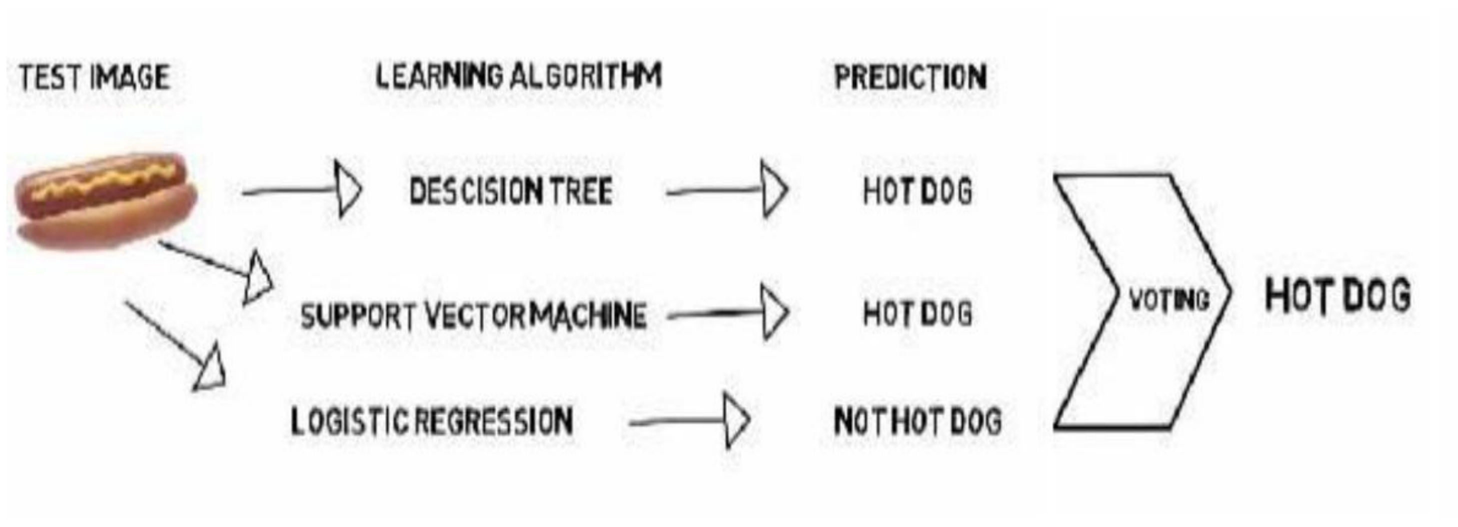
Bagging


Step1: Create Multiple Datasets
D1
D2
...
Dt
Step2: Build Multiple Classifiers
C1
C2
Ct
Step3: Combine Classifiers
C*

How It Works

- ❑ Here all of the learning algorithms may be same or different
- ❑ Decision tree is used widely as the learning algorithm for bagging technique
- ❑ The training dataset can be built with the replacement in the actual dataset
- ❑ Number of the training dataset for each model will be same

- First, the dataset will be divided in to multiple sub datasets.
- Now for each of those sub datasets there will be same or different classification algorithm and for each cases we will get the output.
- After that, there will be an   aggregation algorithm(widely used voting algorithm) which will aggregate all the outputs in to single and the final output produced.


TEST IMAGE
LEARNING ALGORITHM
PREDICTION
DESCISION TREE
HOT DOG
SUPPORT VECTOR MACHINE
HOT DOG
LOGISTIC REGRESSION
NOT HOT DOG
VOTING
HOT DOG

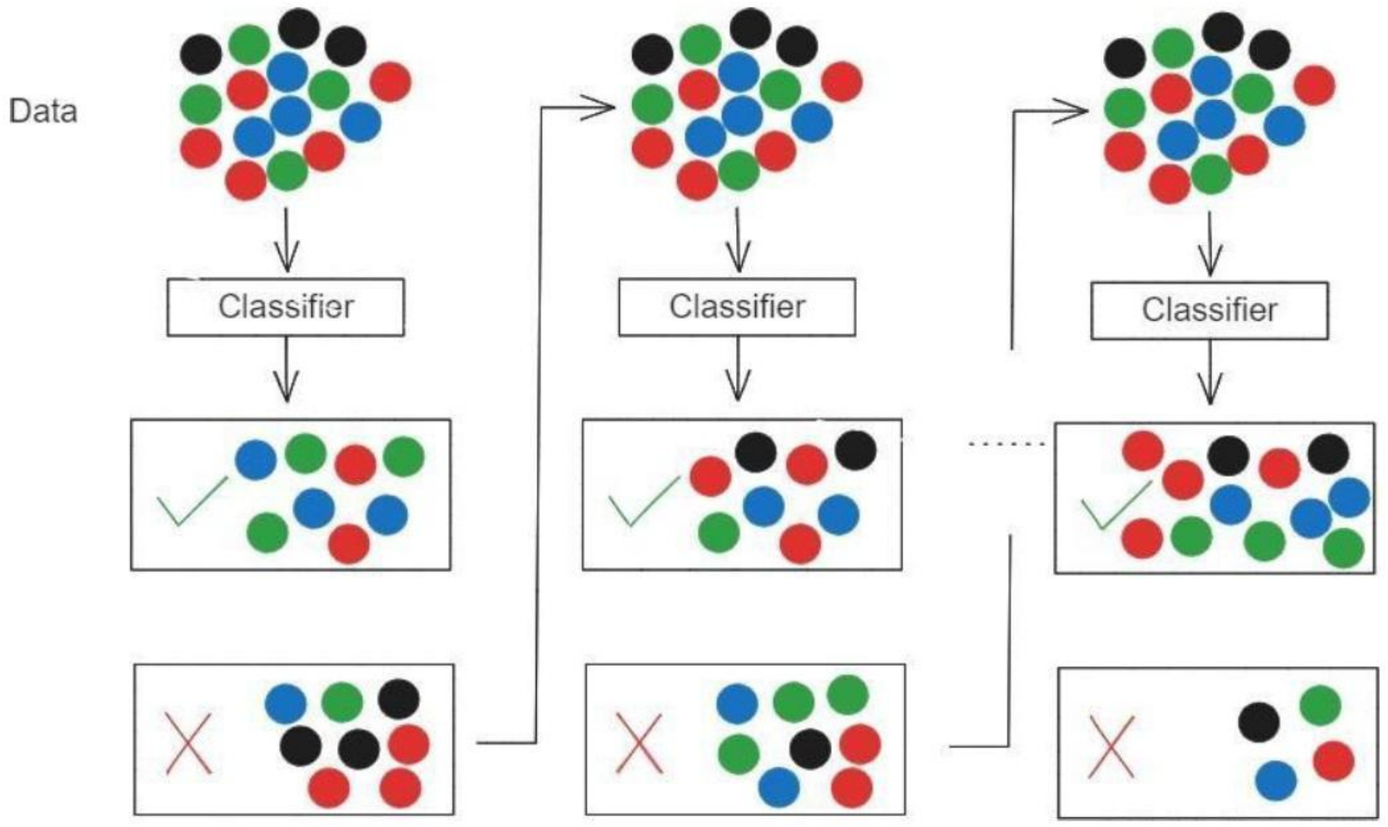
Boosting Model

Models that are typically used in Boosting technique are:

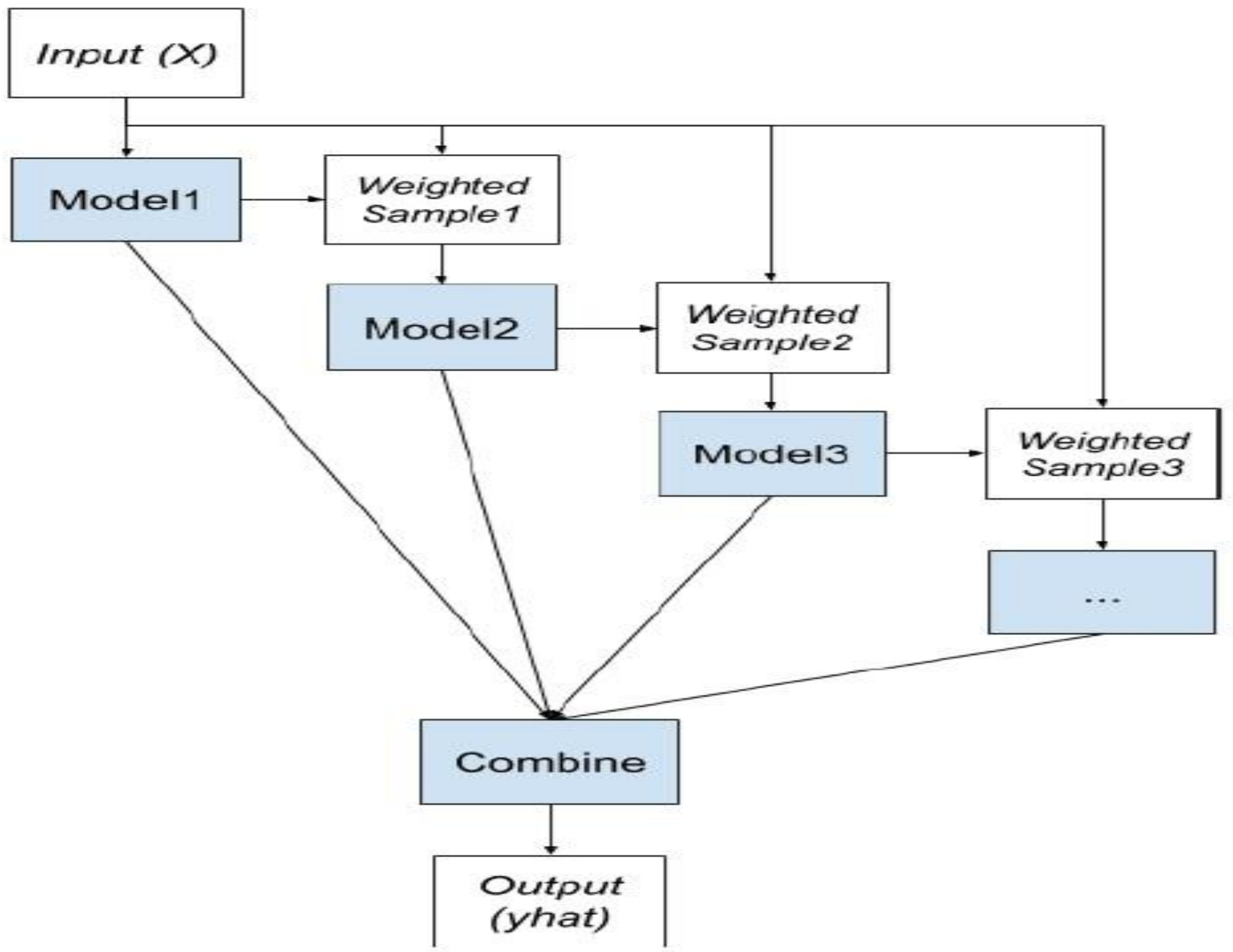
- ❑ XGBoost(Extreme Gradient Boosting)
- ❑ GBM (Gradient Boosting Machine)
- ❑ ADABoost(Adaptive Boosting)

How it works?

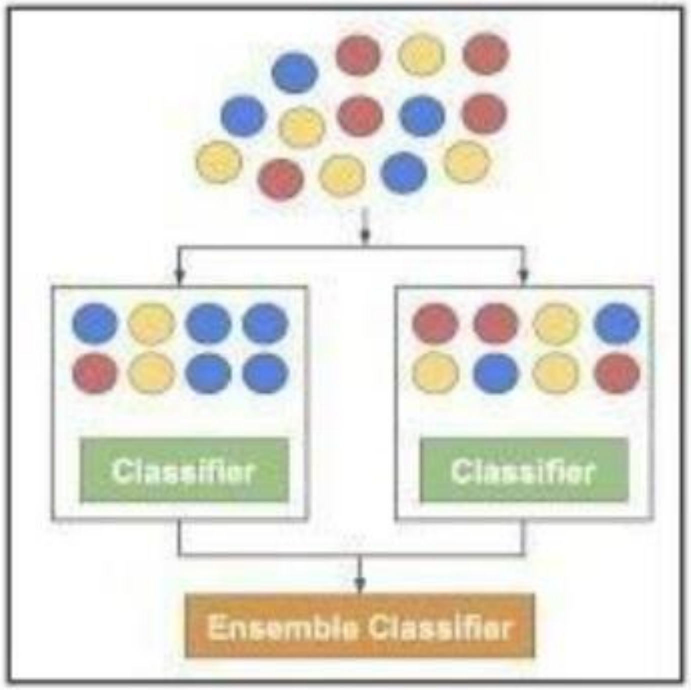
- Here the training will be done in the sequential manner
- First the model will be trained on the whole dataset and while testing there will be some amount of wrongly predicted data. On basis of that, all the wrongly predicted data will be given some weights.
- On the next time , the next classifier will be trained on basis of the full dataset but the correctly predicted data will not be picked again.
- It will keep going until there will no data wrongly predicted or maximum level of training has been achieved.

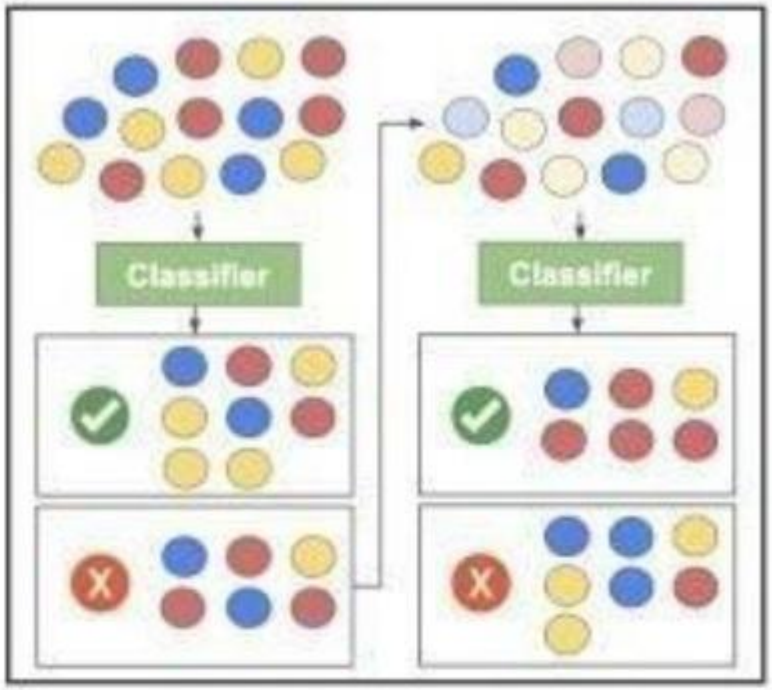

Data
Classifier
Classifier
Classifier
......

Boosting Ensemble



Bagging Vs. Boosting





Problem on Ensemble Classifier

Age	Weight	Diabetics
27	66	1
26	65	0
28	70	0
31	73	1
30	59	1
25	68	???

- ❑ Refer to the table given to the first slide. Apply bagging ensemble classification technique to find out the ultimate output of the final row?

  0: The patient has no diabetics

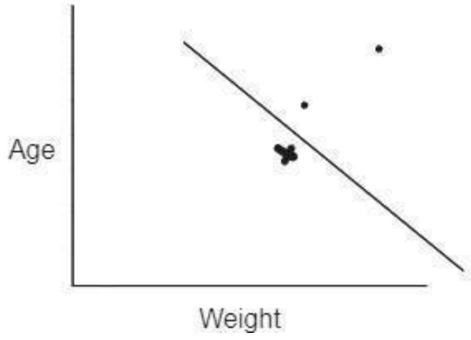
  1: The patient has diabetics
- ❑ Consider 3 classifiers, KNN (with K=1), Linear classifier, Decision Tree Classifier. Take training dataset with 3 rows.

KNN

Age	Weight	Diabetics
26	65	0
28	70	0
30	59	1

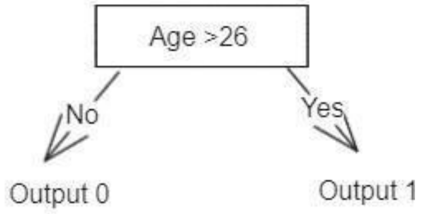
Linear classifier

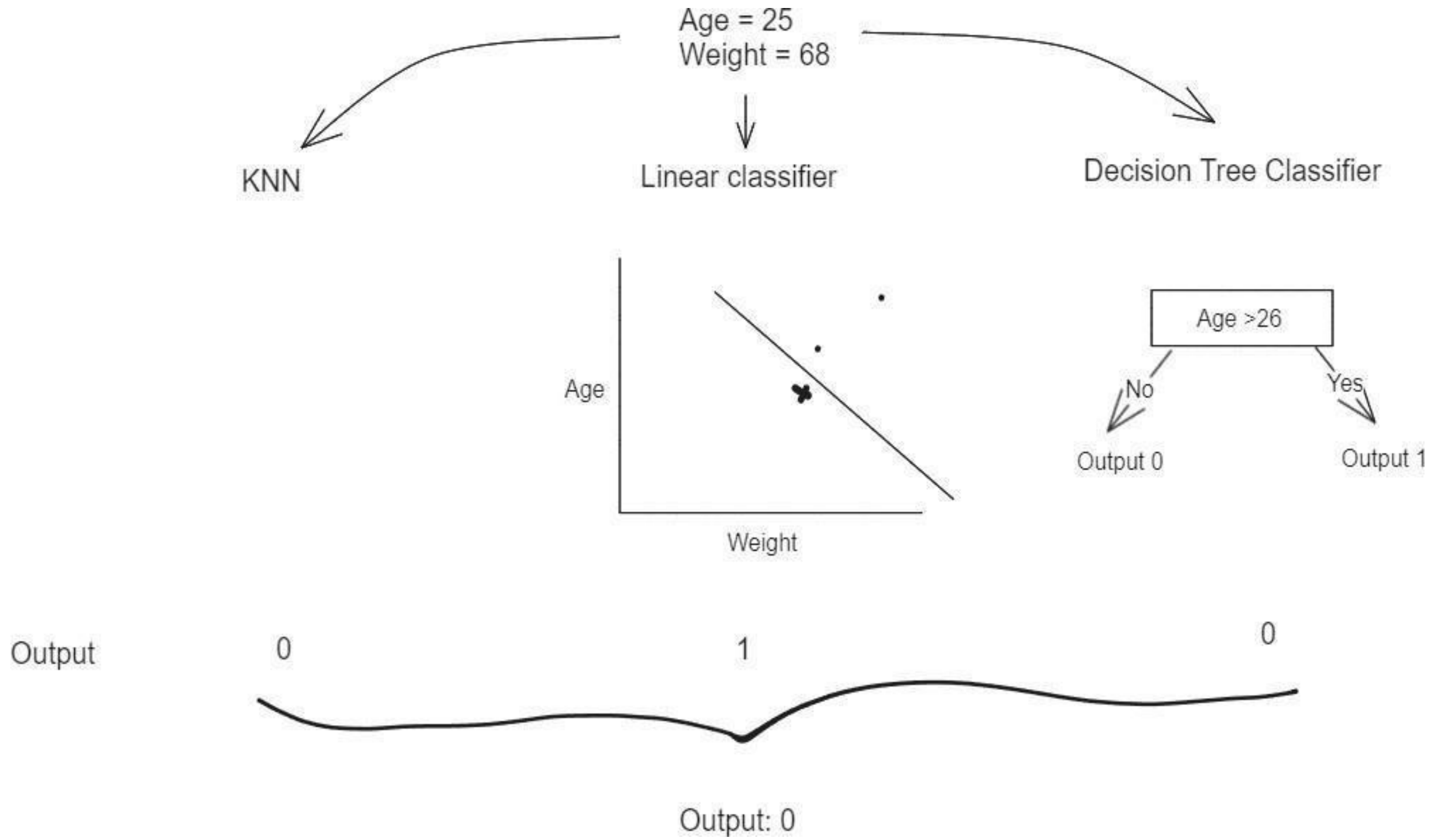
Age	Weight	Diabetics
31	73	1
26	65	0
27	66	1



Decision Tree Classifier

Age	Weight	Diabetics
26	65	0
26	65	0
30	59	1



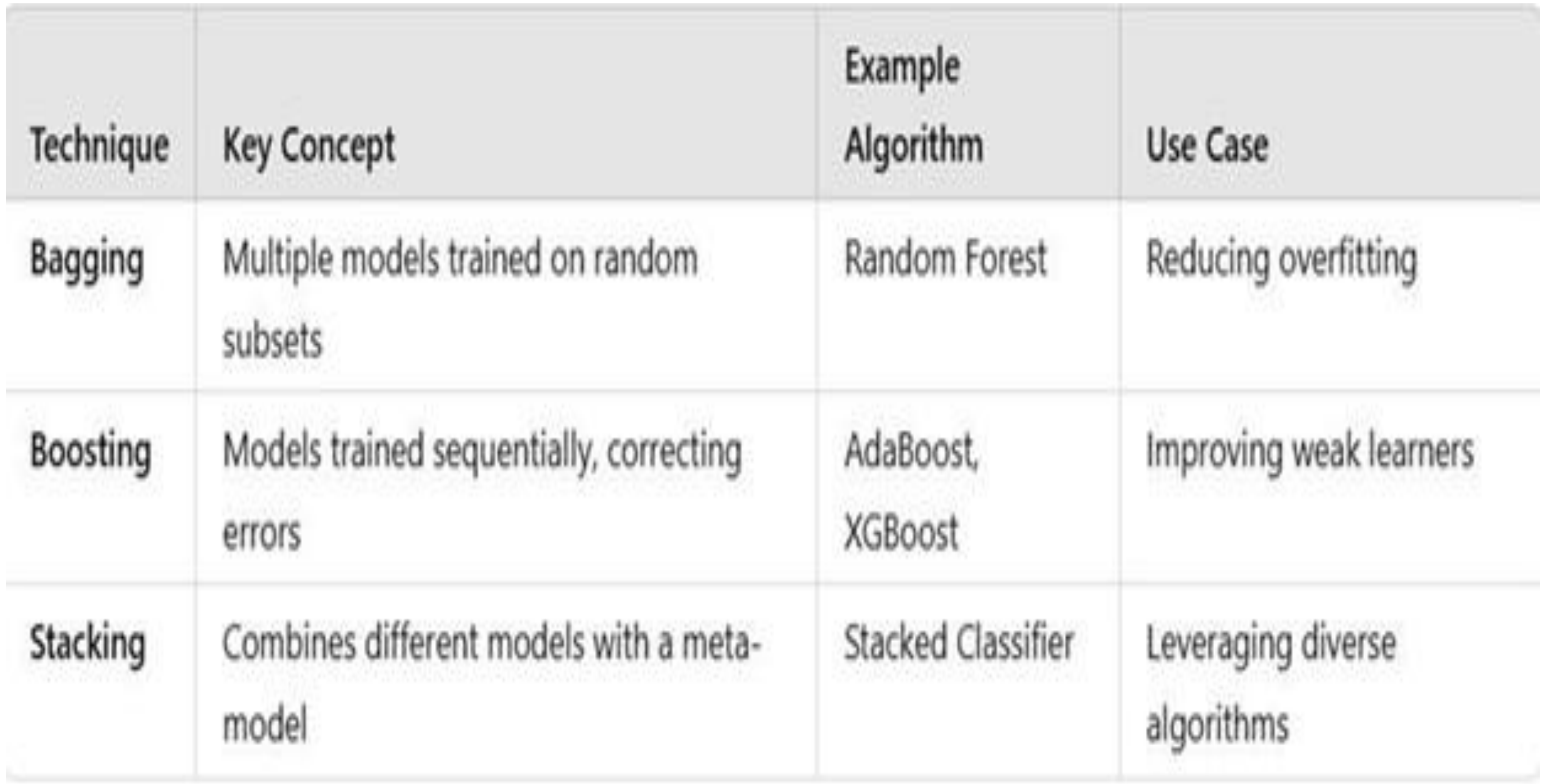

Age = 25
Weight = 68
KNN
Linear classifier
Decision Tree Classifier
Age
Weight
Age >26
No
Yes
Output 0
Output 1
Output
0
1
0
Output: 0

Advantages

- Bagging : Reduces Variance and prevents Overfitting

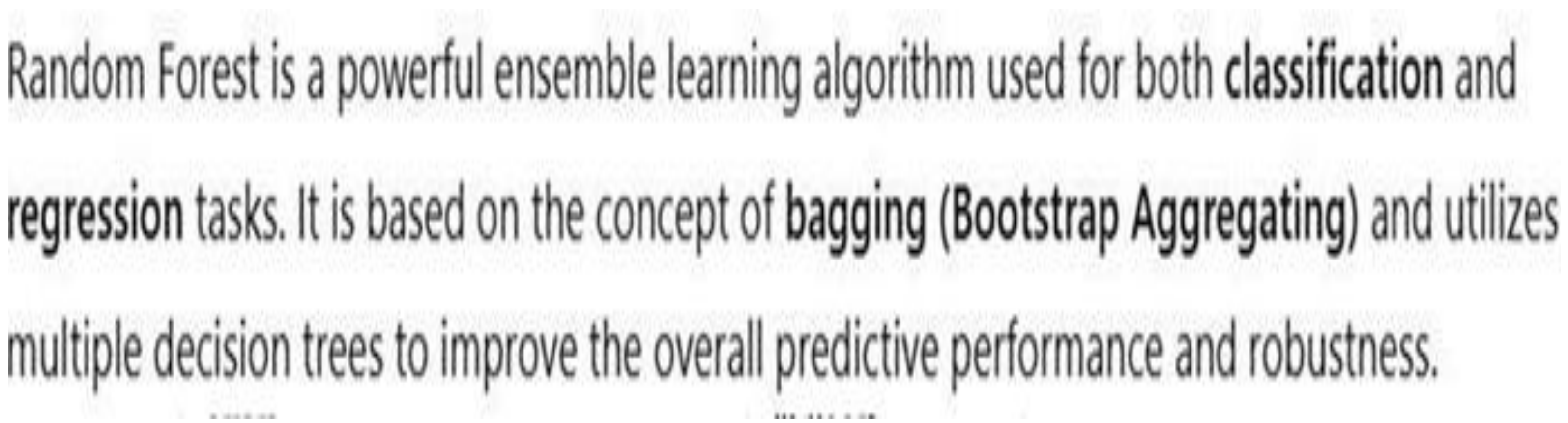
- Boosting : Improves weak learner by iteratively correcting the mistakes

Comparison of Ensemble Techniques

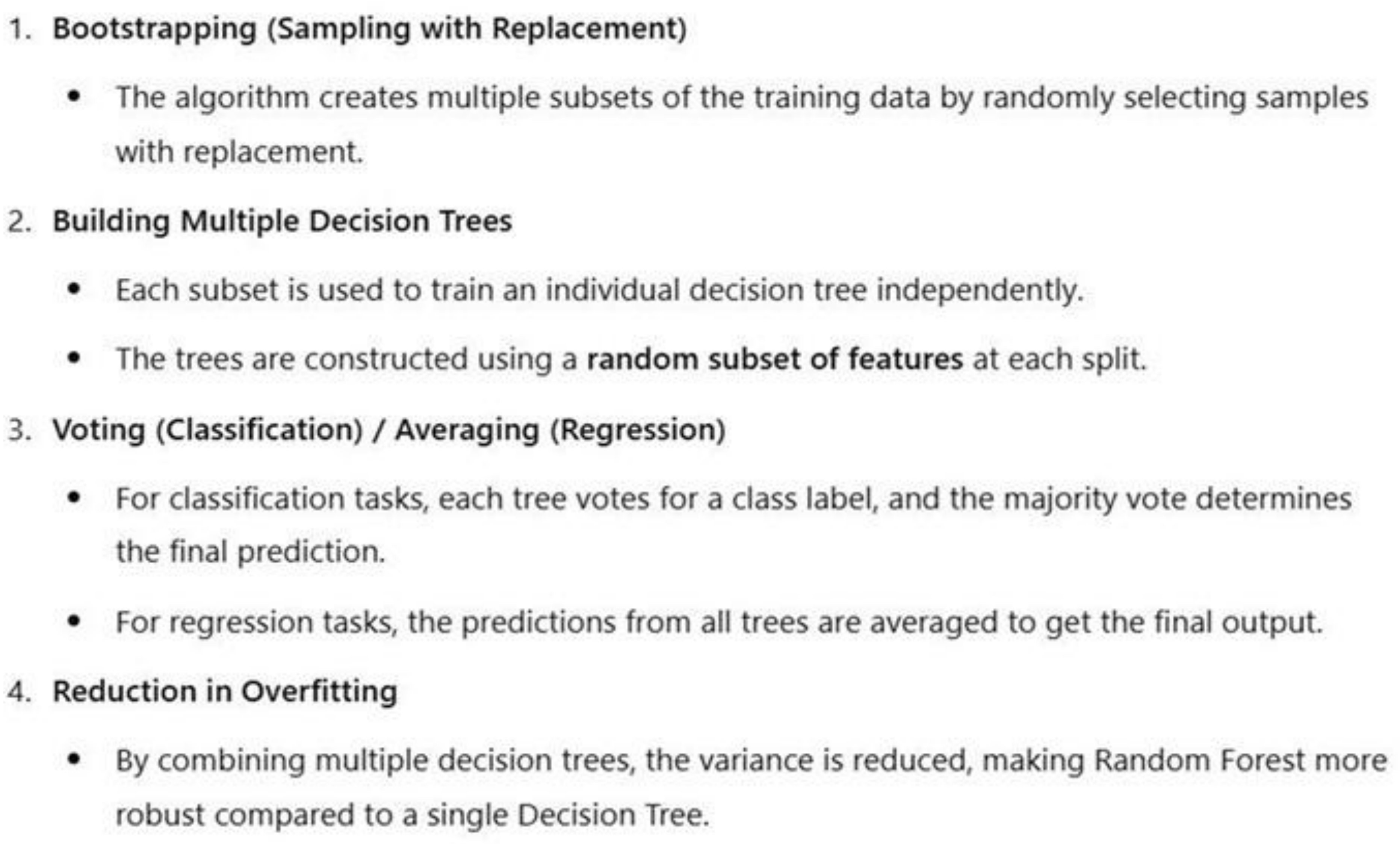


Technique	Key Concept	Example Algorithm	Use Case
Bagging	Multiple models trained on random subsets	Random Forest	Reducing overfitting
Boosting	Models trained sequentially, correcting errors	AdaBoost, XGBoost	Improving weak learners
Stacking	Combines different models with a meta-model	Stacked Classifier	Leveraging diverse algorithms

Random Forest


Random Forest is a powerful ensemble learning algorithm used for both **classification** and **regression** tasks. It is based on the concept of **bagging (Bootstrap Aggregating)** and utilizes multiple decision trees to improve the overall predictive performance and robustness.

How It Works?



1. **Bootstrapping (Sampling with Replacement)**
   - The algorithm creates multiple subsets of the training data by randomly selecting samples with replacement.
2. **Building Multiple Decision Trees**
   - Each subset is used to train an individual decision tree independently.
   - The trees are constructed using a **random subset of features** at each split.
3. **Voting (Classification) / Averaging (Regression)**
   - For classification tasks, each tree votes for a class label, and the majority vote determines the final prediction.
   - For regression tasks, the predictions from all trees are averaged to get the final output.
4. **Reduction in Overfitting**
   - By combining multiple decision trees, the variance is reduced, making Random Forest more robust compared to a single Decision Tree.

Explanation of the Illustration

We demonstrated **Random Forest** using two real-world examples:

1. **Loan Default Prediction (Classification)**
2. **House Price Prediction (Regression)**

Both models were built using synthetic datasets, trained on real-world-inspired features, and evaluated using appropriate performance metrics.

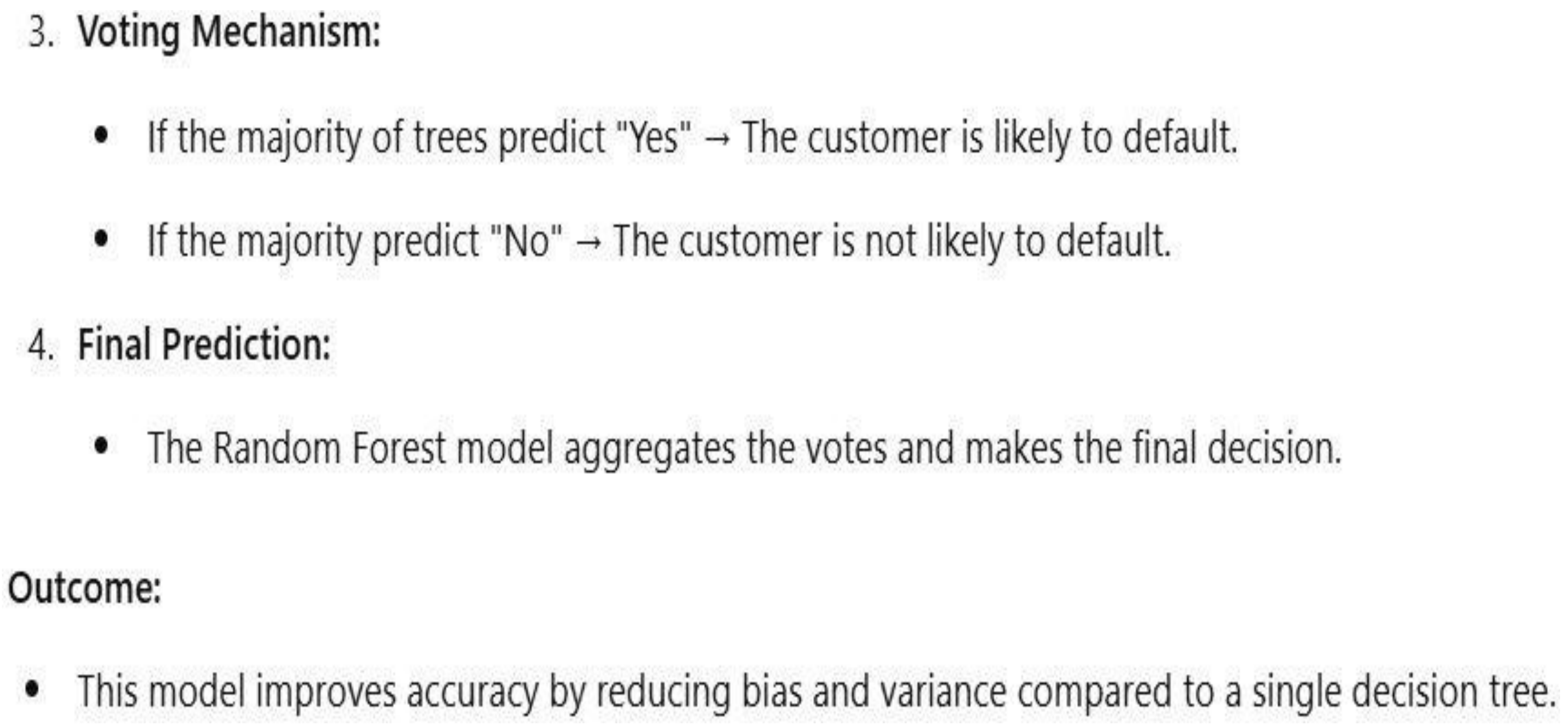
Problem Statement

A bank wants to predict whether a customer will default on a loan based on various attributes such as **income, credit score, loan amount, and employment status.**

Steps:

1. **Prepare the dataset:**
   - Features: Income, Credit Score, Loan Amount, Employment Status
   - Target: **Default (Yes/No)**
2. **Train the Random Forest Model:**
   - Generate multiple decision trees using different feature subsets.
   - Each tree predicts whether the customer will default.
3. **Voting Mechanism:**
   - If the majority of trees predict "Yes" → The customer is likely to default.
   - If the majority predict "No" → The customer is not likely to default.

Example I: Random Forest(Classification)



3. **Voting Mechanism:**
   - If the majority of trees predict "Yes" → The customer is likely to default.
   - If the majority predict "No" → The customer is not likely to default.
4. **Final Prediction:**
   - The Random Forest model aggregates the votes and makes the final decision.

Outcome:

- This model improves accuracy by reducing bias and variance compared to a single decision tree.

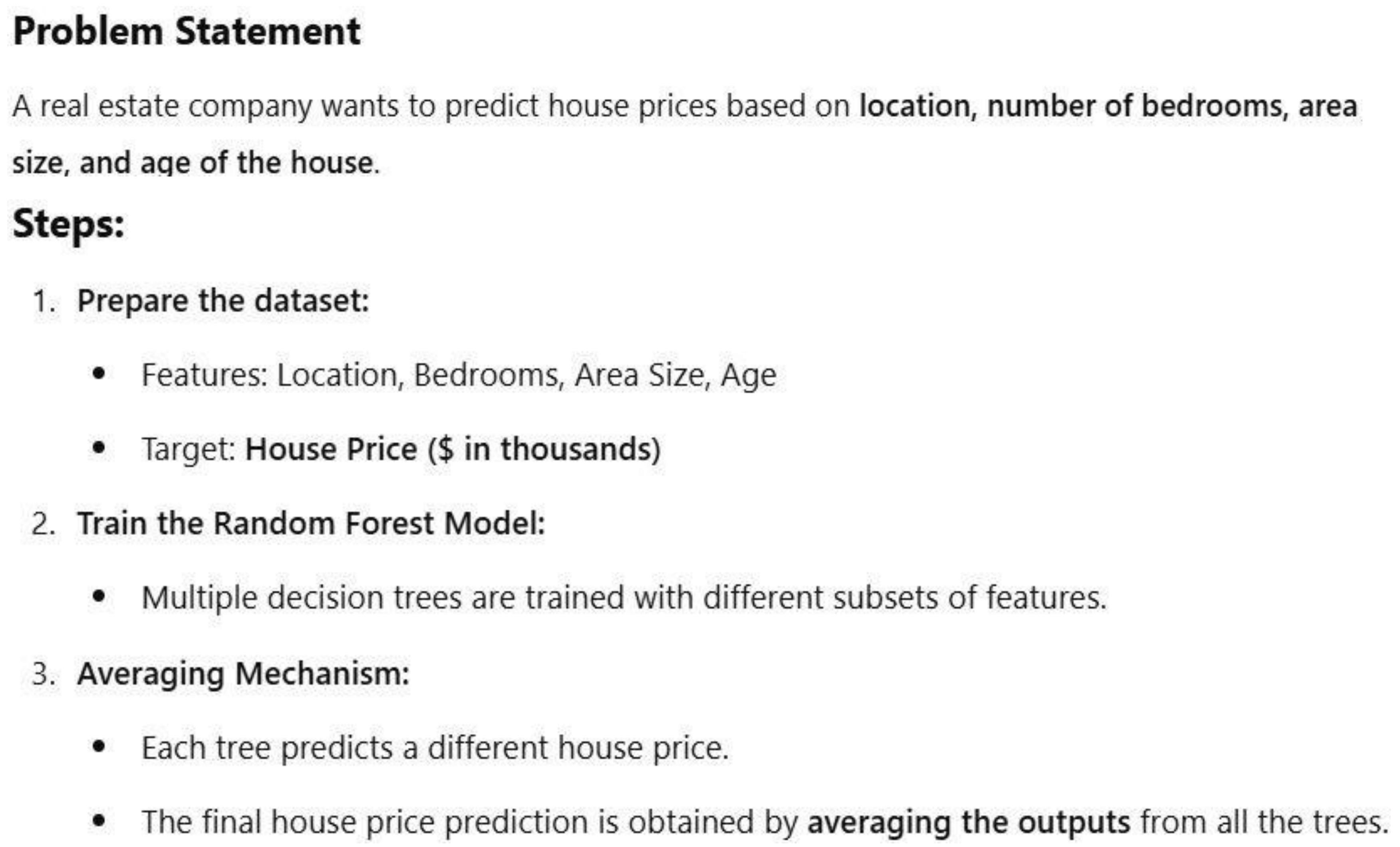
- Created a dataset with **500 records** and features:
  - **Income:** Annual income in dollars.
  - **Credit Score:** Ranges from 300 to 850.
  - **Loan Amount:** Loan amount in dollars.
  - **Employment Status:** (0 = Unemployed, 1 = Employed)
  - **Target Variable (Default):** (0 = No Default, 1 = Default)

2. **Data Splitting:**
    - Split data into **80% training** and **20% testing** sets.
3. **Training the Random Forest Classifier:**
    - Used **100 decision trees** (estimators).
    - Each tree makes a classification, and the majority vote determines the final prediction.
4. **Prediction & Evaluation:**
    - The model predicts whether a customer will default.
    - **Accuracy Score** is computed to measure performance.

Result:

- The accuracy score indicates how well the model predicts loan defaults.
- Higher accuracy means the model is better at distinguishing between defaulters and non-defaulters.

Example II: Random Forest (Regression)



Problem Statement

A real estate company wants to predict house prices based on **location, number of bedrooms, area size, and age of the house.**

Steps:

1. **Prepare the dataset:**
   - Features: Location, Bedrooms, Area Size, Age
   - Target: **House Price ($ in thousands)**
2. **Train the Random Forest Model:**
   - Multiple decision trees are trained with different subsets of features.
3. **Averaging Mechanism:**
   - Each tree predicts a different house price.
   - The final house price prediction is obtained by **averaging the outputs** from all the trees.

4. **Final Prediction:**
   - Suppose different trees predict:
     - $250K, $270K, $260K, $255K, $265K
   - The final prediction = **(250+270+260+255+265)/5 = $260K**

Outcome:

- The model provides a more reliable estimate by averaging predictions, reducing the risk of outliers.

Advantages of Random Forest

✔ **Handles Missing Values** – Works well even with incomplete data.

✔ **Reduces Overfitting** – Combines multiple trees to generalize better.

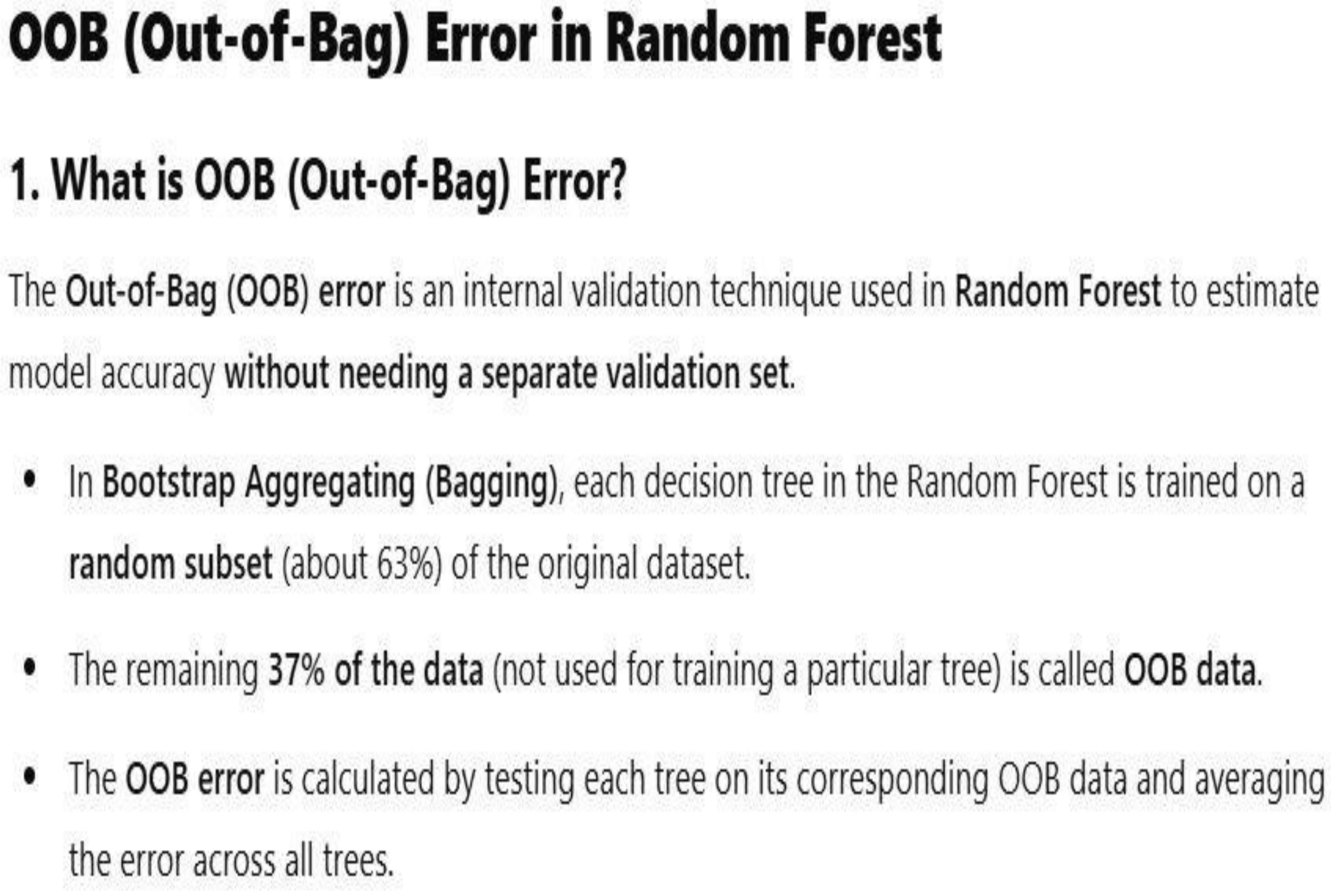
✔ **Handles Large Datasets** – Works well with high-dimensional data.

✔ **Feature Importance** – Helps identify which features are most important.

Disadvantages of Random Forest

❌ **Computationally Expensive** – Requires more processing power compared to single decision trees.

❌ **Less Interpretability** – Harder to interpret compared to a single decision tree.



OOB (Out-of-Bag) Error in Random Forest

1. What is OOB (Out-of-Bag) Error?

The **Out-of-Bag (OOB) error** is an internal validation technique used in **Random Forest** to estimate model accuracy **without needing a separate validation set**.

- In **Bootstrap Aggregating (Bagging)**, each decision tree in the Random Forest is trained on a **random subset** (about 63%) of the original dataset.
- The remaining **37% of the data** (not used for training a particular tree) is called **OOB data**.
- The **OOB error** is calculated by testing each tree on its corresponding OOB data and averaging the error across all trees.

2. How is OOB Error Calculated?

1. **For each tree in the Random Forest**, a random subset (about 63% of the training data) is drawn **with replacement**.
2. The remaining 37% of the data (**OOB samples**) are **not used** to train that tree.
3. Each tree makes predictions on its respective OOB samples.
4. The **OOB error** is computed as the average error across all trees for their OOB samples.

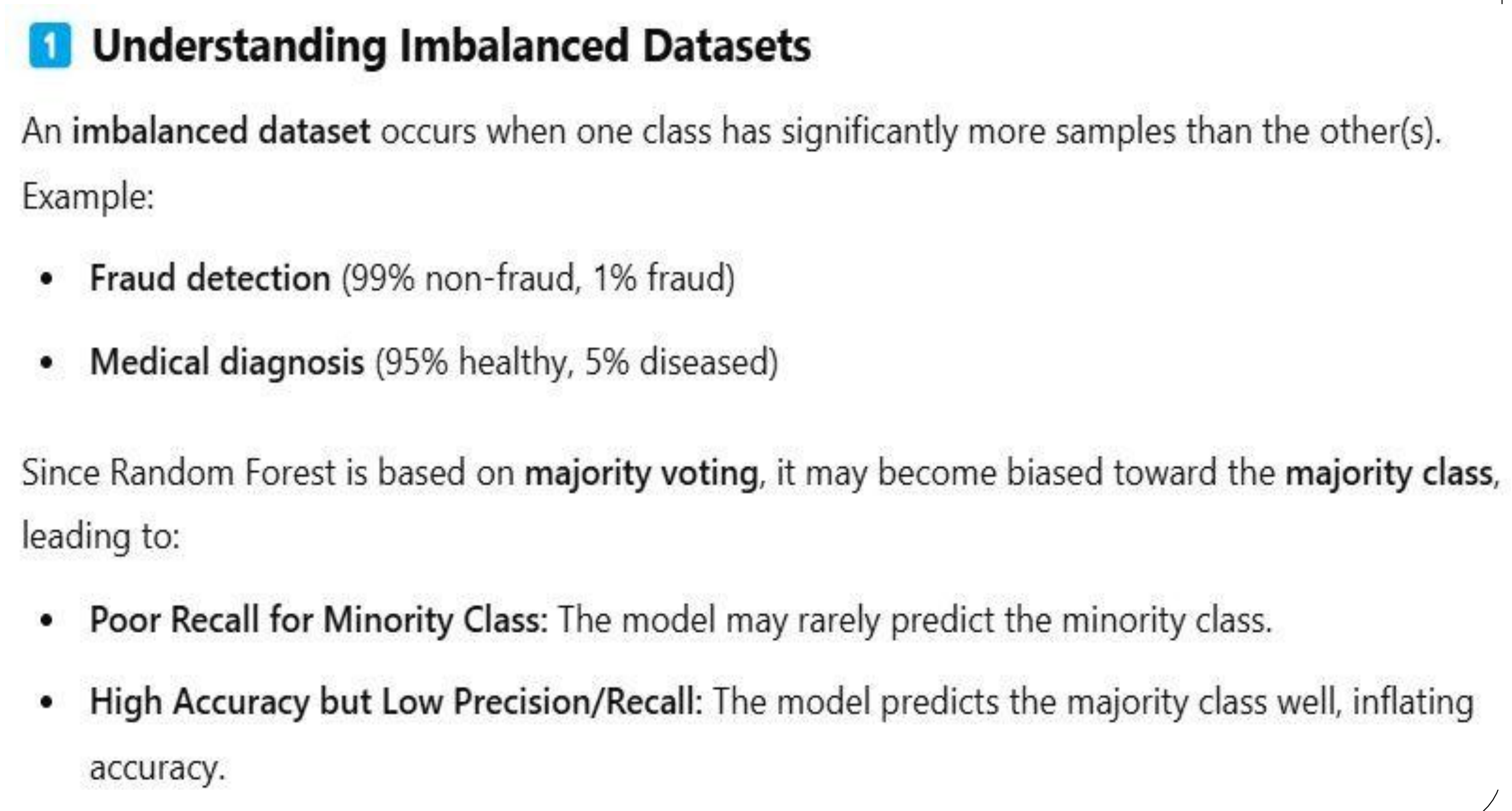
Mathematically:

$$OOB\_Error = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{y}_{OOB})$$

where:

- N = total number of samples
- y_i = true label of sample i
- $\hat{y}_{OOB}$ = aggregated OOB predictions for sample i
- L = loss function (e.g., misclassification rate for classification, MSE for regression)

How Does Random Forest Handle Imbalanced Datasets?



1️⃣ Understanding Imbalanced Datasets

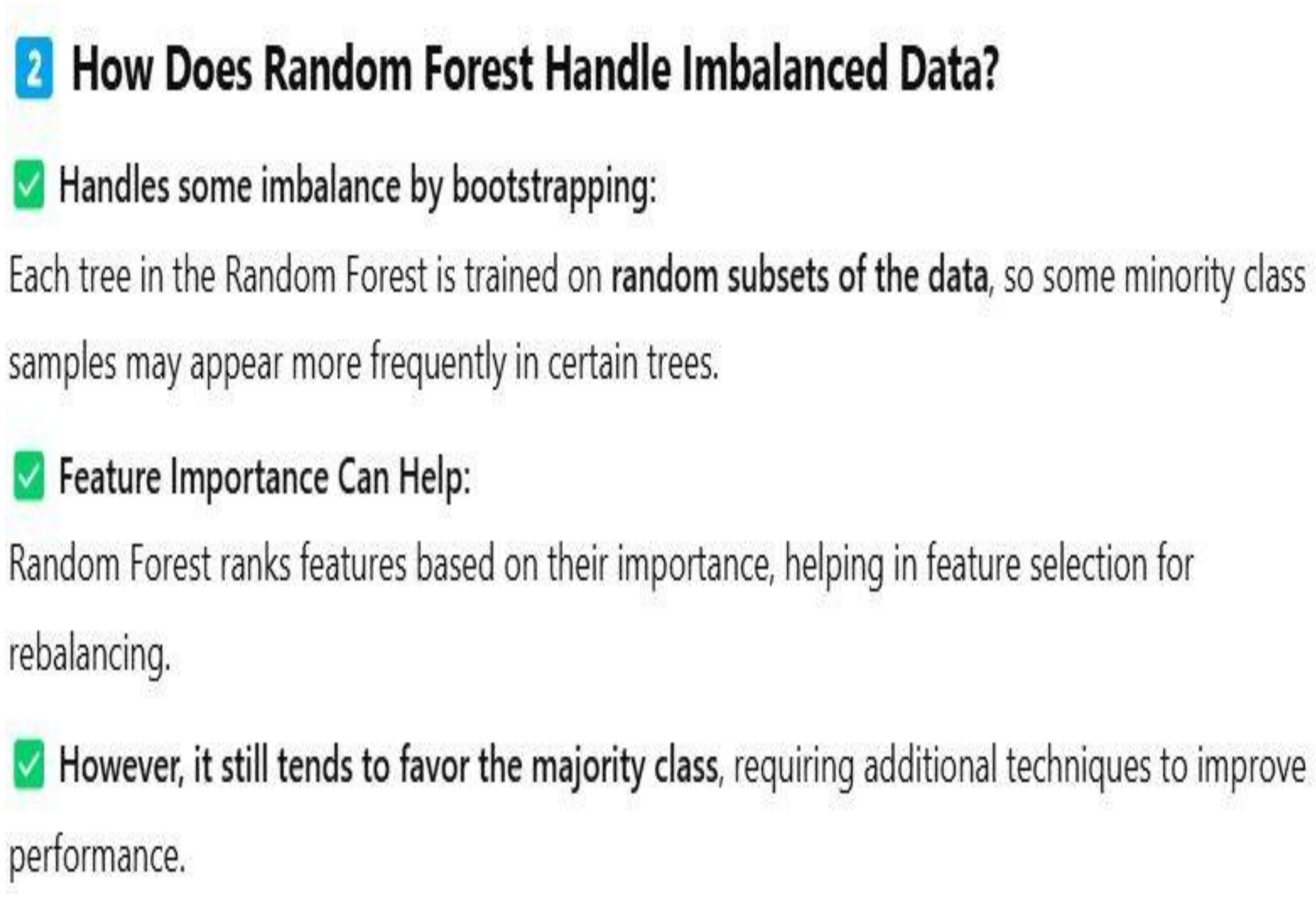
An **imbalanced dataset** occurs when one class has significantly more samples than the other(s).

Example:

- **Fraud detection** (99% non-fraud, 1% fraud)
- **Medical diagnosis** (95% healthy, 5% diseased)

Since Random Forest is based on **majority voting**, it may become biased toward the **majority class**, leading to:

- **Poor Recall for Minority Class:** The model may rarely predict the minority class.
- **High Accuracy but Low Precision/Recall:** The model predicts the majority class well, inflating accuracy.



2️⃣ How Does Random Forest Handle Imbalanced Data?

✅ Handles some imbalance by bootstrapping:

Each tree in the Random Forest is trained on **random subsets of the data**, so some minority class samples may appear more frequently in certain trees.

✅ Feature Importance Can Help:

Random Forest ranks features based on their importance, helping in feature selection for rebalancing.

✅ **However, it still tends to favor the majority class**, requiring additional techniques to improve performance.

📌 1. Class Weight Adjustment (Built-in Method)

- Random Forest allows assigning **higher weights** to the minority class.
- The model prioritizes the minority class during training.

📌 2. Oversampling the Minority Class

- Increases the number of minority class samples by duplicating existing ones or generating synthetic data.

🔹 Method 1: Random Oversampling

Simply duplicates random samples from the minority class.

🔹 Method 2: SMOTE (Synthetic Minority Over-sampling Technique)

- Generates synthetic examples instead of simple duplication.
- Creates "new" samples based on existing data points.

📌 3. Undersampling the Majority Class

- Reduces the number of samples from the majority class to balance the dataset.

🔹 **Method 1: Random Undersampling**

- Removes random samples from the majority class.
- May lose useful information.

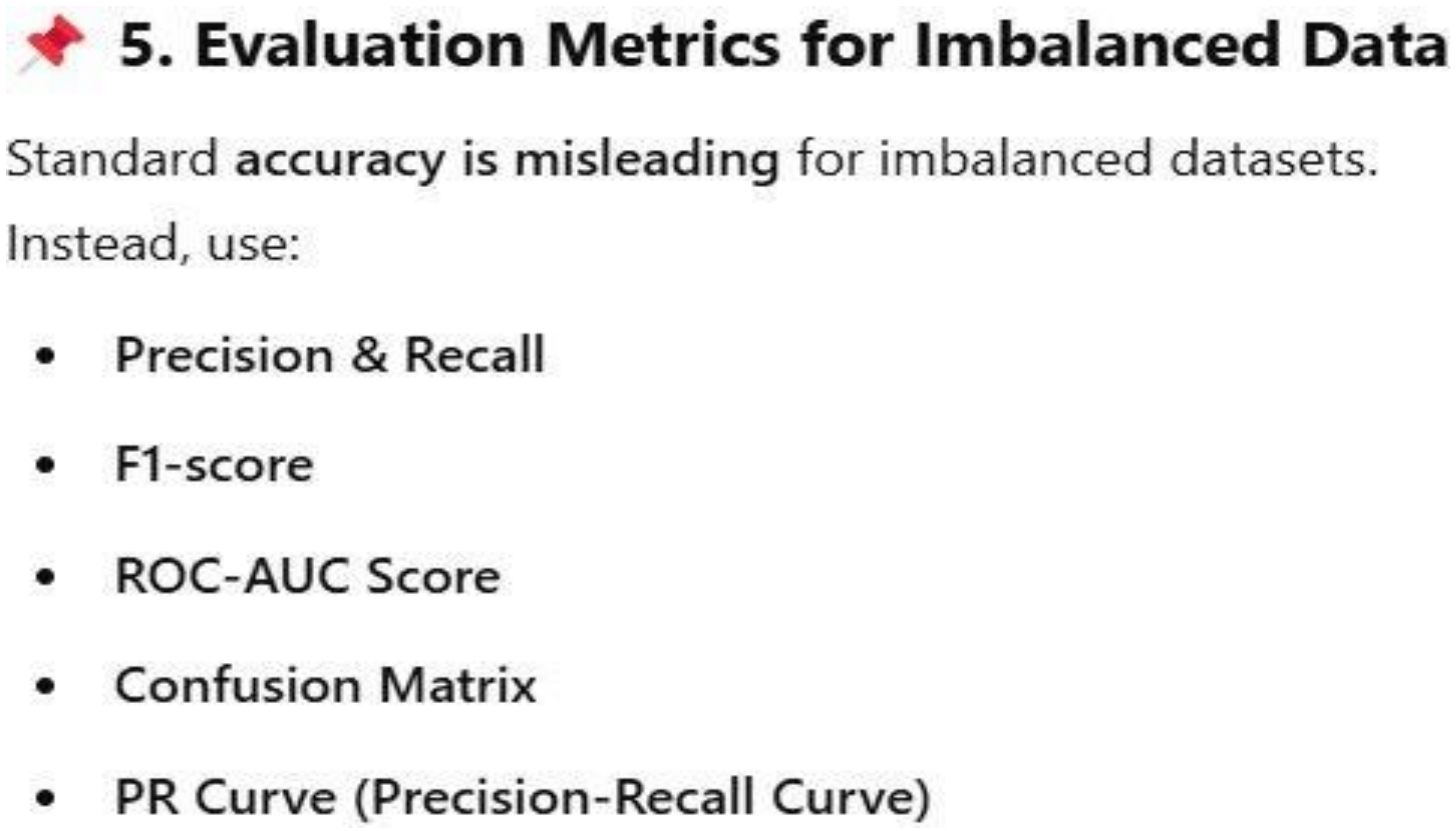
🔹 **Method 2: NearMiss Algorithm**

- Selects majority class samples **closest** to the minority class instead of random removal.

📌 4. Ensemble Methods for Imbalanced Data

🔹 **Balanced Random Forest**

- A variant of Random Forest that **undersamples** the majority class **within each tree.**



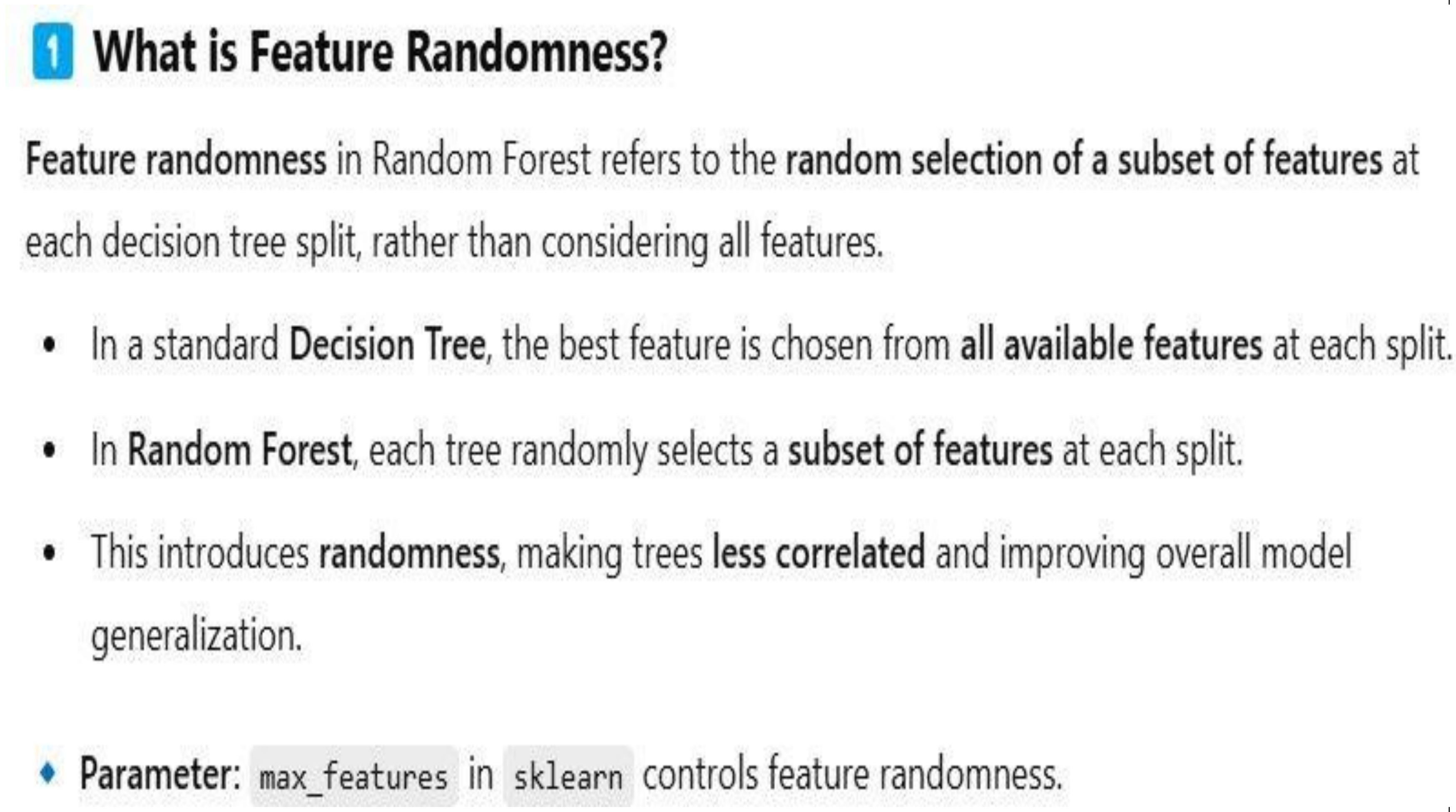
📌 5. Evaluation Metrics for Imbalanced Data

Standard **accuracy is misleading** for imbalanced datasets.

Instead, use:

- **Precision & Recall**
- **F1-score**
- **ROC-AUC Score**
- **Confusion Matrix**
- **PR Curve (Precision-Recall Curve)**

Ensemble Learning: Feature Randomness in Random Forest

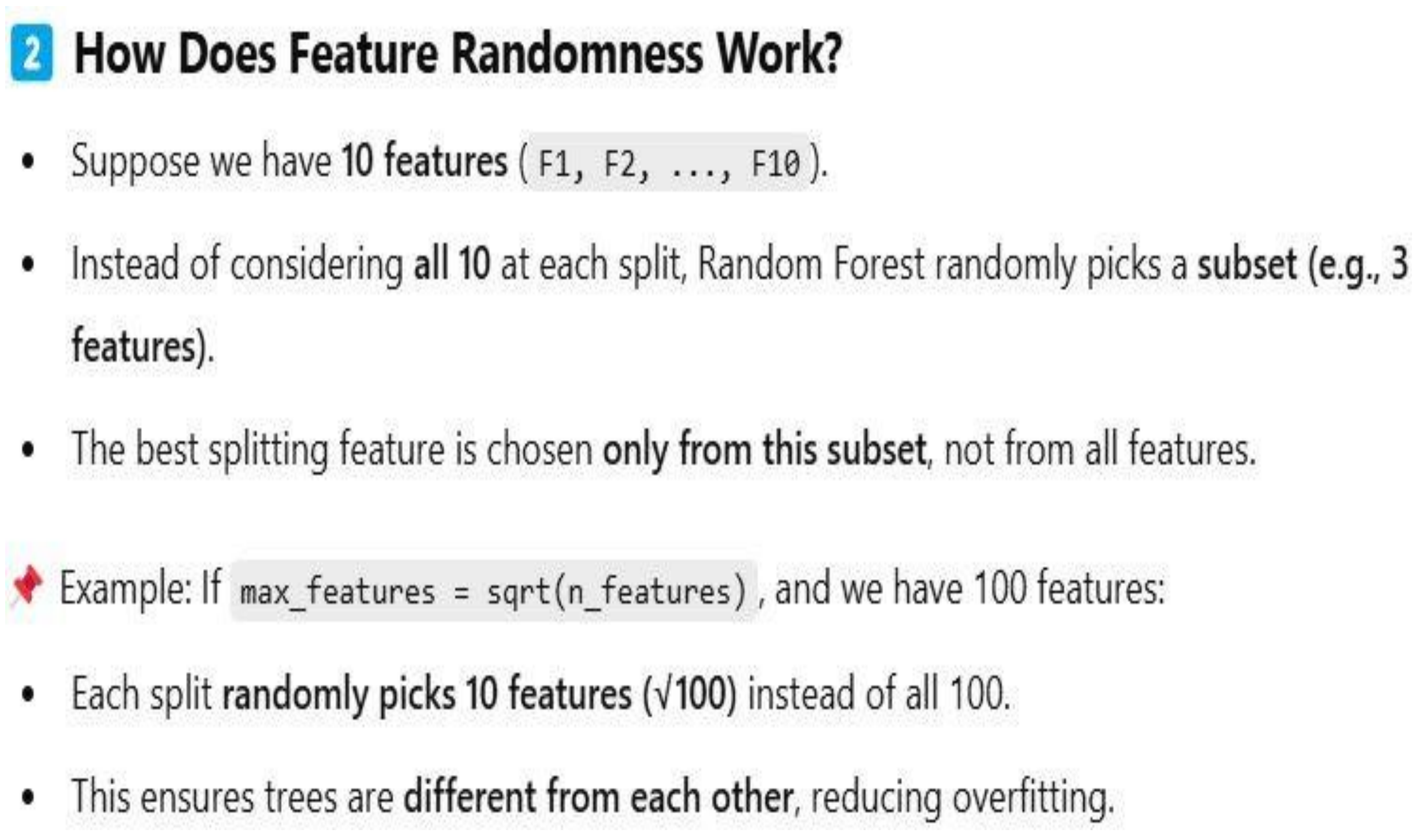


1 What is Feature Randomness?

Feature randomness in Random Forest refers to the **random selection of a subset of features** at each decision tree split, rather than considering all features.

- In a standard **Decision Tree**, the best feature is chosen from **all available features** at each split.
- In **Random Forest**, each tree randomly selects a **subset of features** at each split.
- This introduces **randomness**, making trees **less correlated** and improving overall model generalization.

◆ **Parameter:** `max_features` in `sklearn` controls feature randomness.



2️⃣ How Does Feature Randomness Work?

- Suppose we have **10 features** (`F1, F2, ..., F10`).
- Instead of considering **all 10** at each split, Random Forest randomly picks a **subset (e.g., 3 features)**.
- The best splitting feature is chosen **only from this subset**, not from all features.

📌 Example: If `max_features = sqrt(n_features)` , and we have 100 features:

- Each split **randomly picks 10 features** (√100) instead of all 100.
- This ensures trees are **different from each other**, reducing overfitting.

3️⃣ Why is Feature Randomness Important?

✅ 1. Reduces Overfitting

- If all trees select splits based on the **same dominant feature**, they become highly correlated.
- Feature randomness forces trees to explore different patterns, making the model more **robust**.

✅ 2. Improves Model Diversity

- Different trees rely on different subsets of features, making the ensemble stronger.
- Helps Random Forest work well **even with noisy or redundant features**.

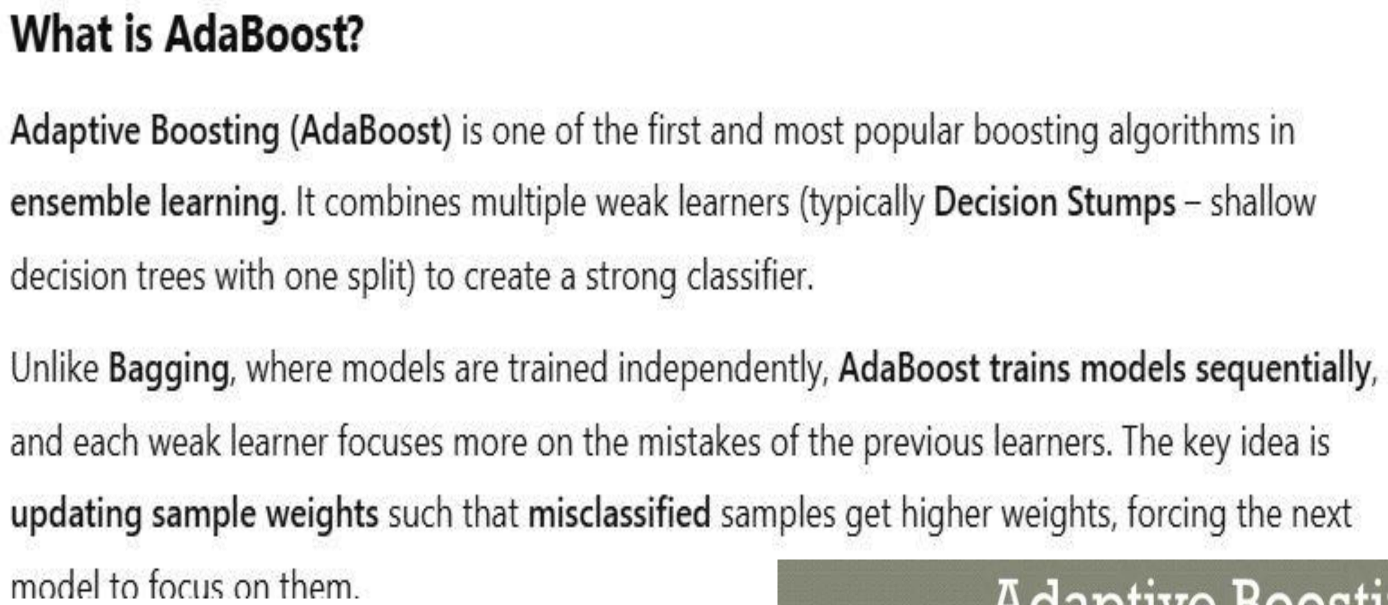
✅ 3. Handles High-Dimensional Data

- In large datasets with many features, computing the best split over **all features** is expensive.
- Limiting the number of considered features speeds up training.

✅ 4. Enhances Generalization

- Trees trained on different feature subsets generalize better to unseen data.

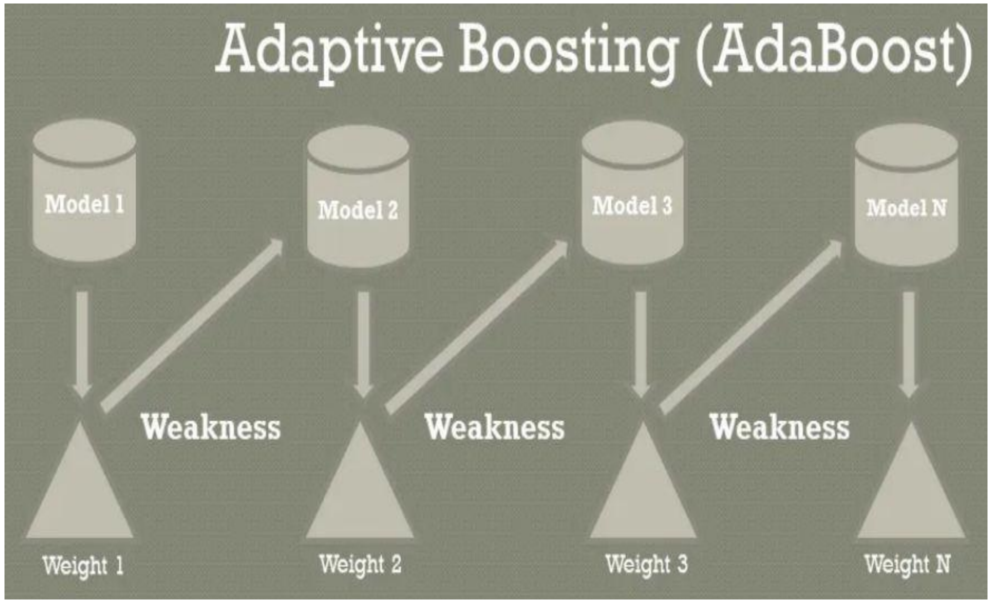
Ensemble Learning: AdaBoost



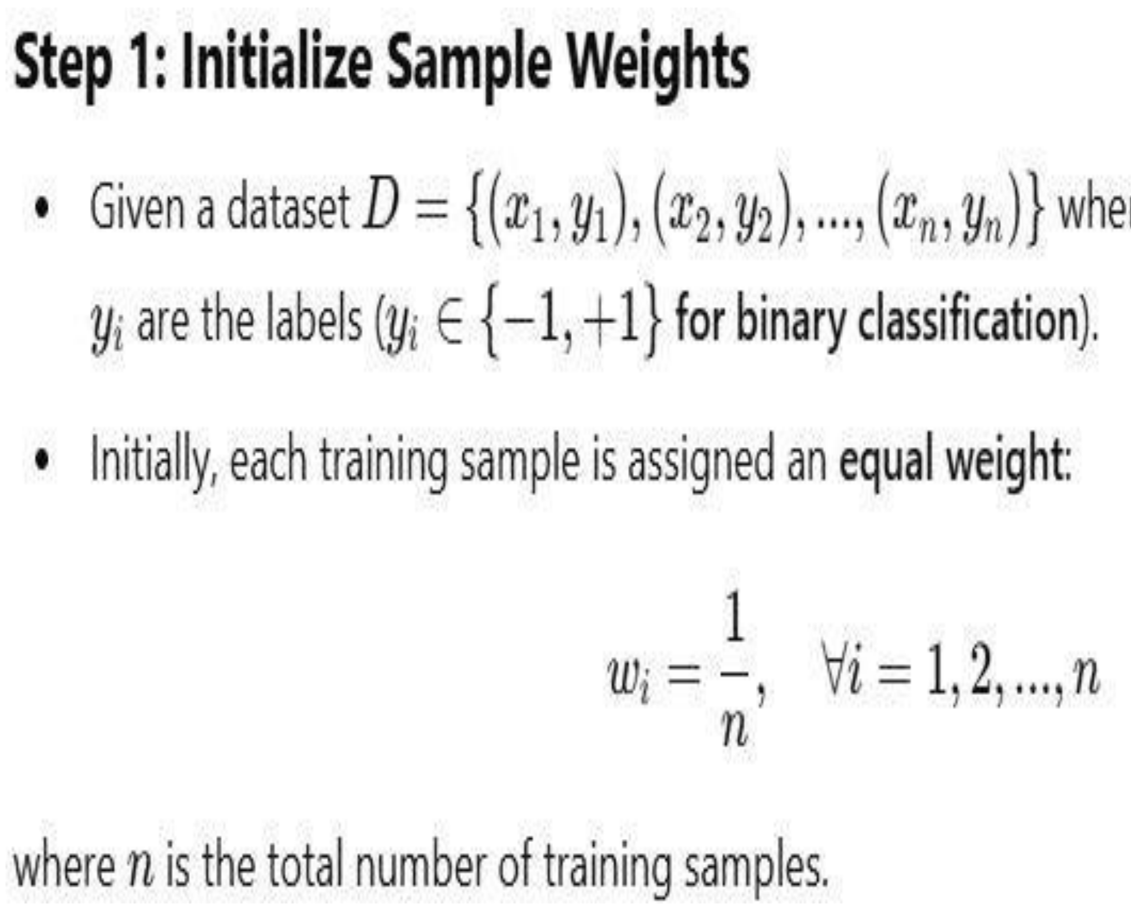
What is AdaBoost?

Adaptive Boosting (AdaBoost) is one of the first and most popular boosting algorithms in **ensemble learning.** It combines multiple weak learners (typically **Decision Stumps** – shallow decision trees with one split) to create a strong classifier.

Unlike **Bagging**, where models are trained independently, **AdaBoost trains models sequentially**, and each weak learner focuses more on the mistakes of the previous learners. The key idea is **updating sample weights** such that **misclassified** samples get higher weights, forcing the next model to focus on them.



How it Works ?



Step 1: Initialize Sample Weights

- Given a dataset $D = \{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ where x_i are the input features and y_i are the labels ($y_i \in \{-1, +1\}$ **for binary classification**).
- Initially, each training sample is assigned an **equal weight**:

$$w_i = \frac{1}{n}, \quad \forall i = 1, 2, ..., n$$

where n is the total number of training samples.

Step 2: Train a Weak Learner

- Train a weak classifier $h_t(x)$ (e.g., a decision stump) using the weighted dataset.
- The weak classifier tries to **minimize the weighted classification error**:

$$e_t = \sum_{i=1}^{n} w_i I(y_i \neq h_t(x_i))$$

where:

- $I(y_i \neq h_t(x_i))$ is an indicator function that returns **1 if misclassified**, else **0**.
- e_t is the total weighted error.

Step 3: Compute Model Weight (α_t)

- Compute the weight of the weak model (α_t), which determines its importance in the final classifier:

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - e_t}{e_t} \right)$$

- If e_t is small (i.e., the classifier is good), then α_t is large, meaning it gets more importance.
- If e_t is large (i.e., the classifier is weak), then α_t is small.

Step 4: Update Sample Weights

- The weights of misclassified samples are **increased**, and correctly classified samples are **decreased** so that the next weak learner focuses more on hard examples.
- Update sample weights:

$$w_i^{(t+1)} = w_i^{(t)} \times \exp(\alpha_t I(y_i \neq h_t(x_i)))$$

- Normalize the weights so that they sum to 1:

$$w_i^{(t+1)} = \frac{w_i^{(t+1)}}{\sum w_i^{(t+1)}}$$

This ensures that misclassified samples get **higher importance** in the next round.

Step 5: Repeat for T Iterations

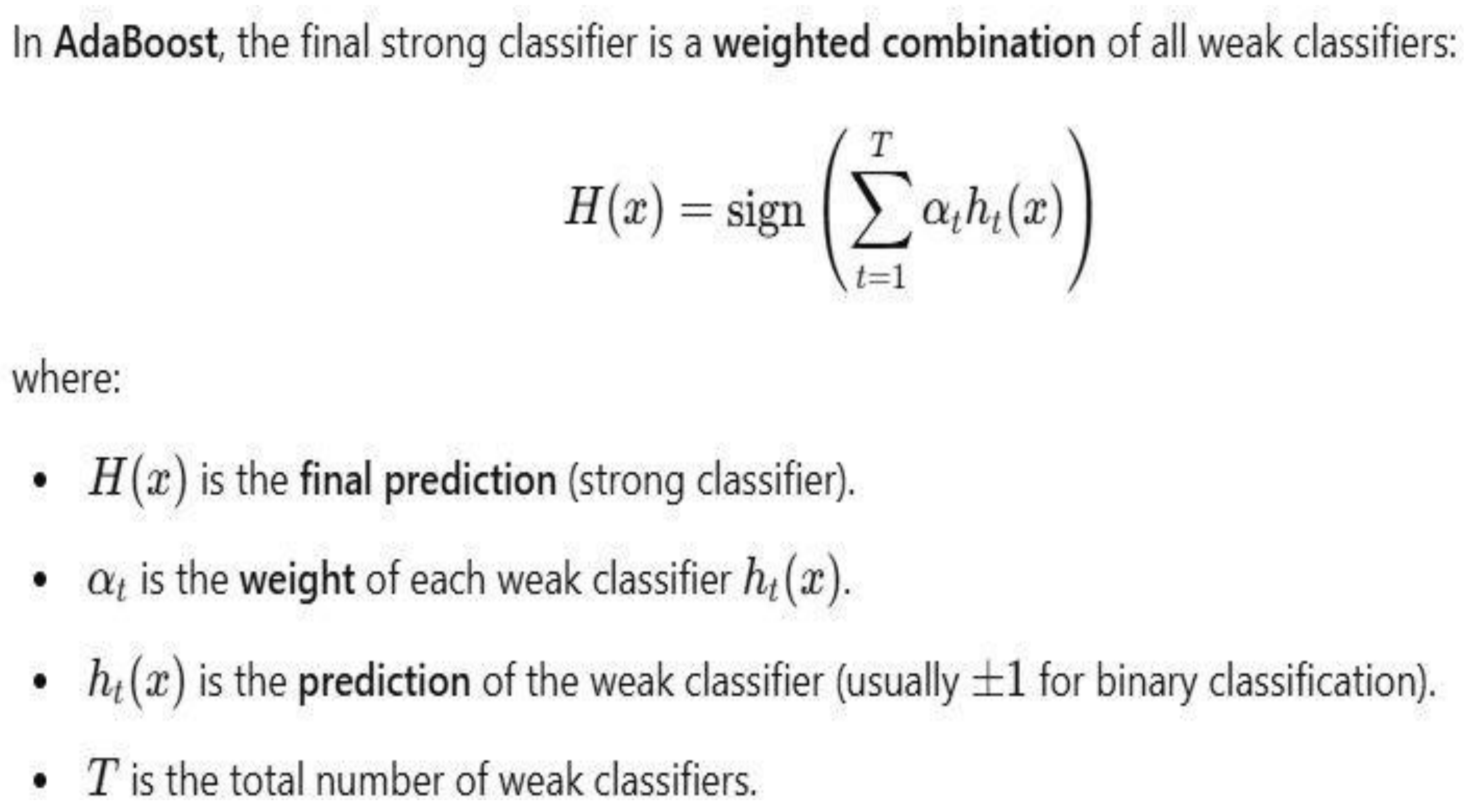
- Repeat **steps 2–4** for T iterations, training multiple weak classifiers and adjusting sample weights in each round.

Step 6: Final Strong Classifier

- The final boosted classifier is a **weighted combination** of all weak classifiers:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

- Each weak learner **votes** for a class, and its vote is weighted by α_t.
- The **sign function** determines the final predicted class.

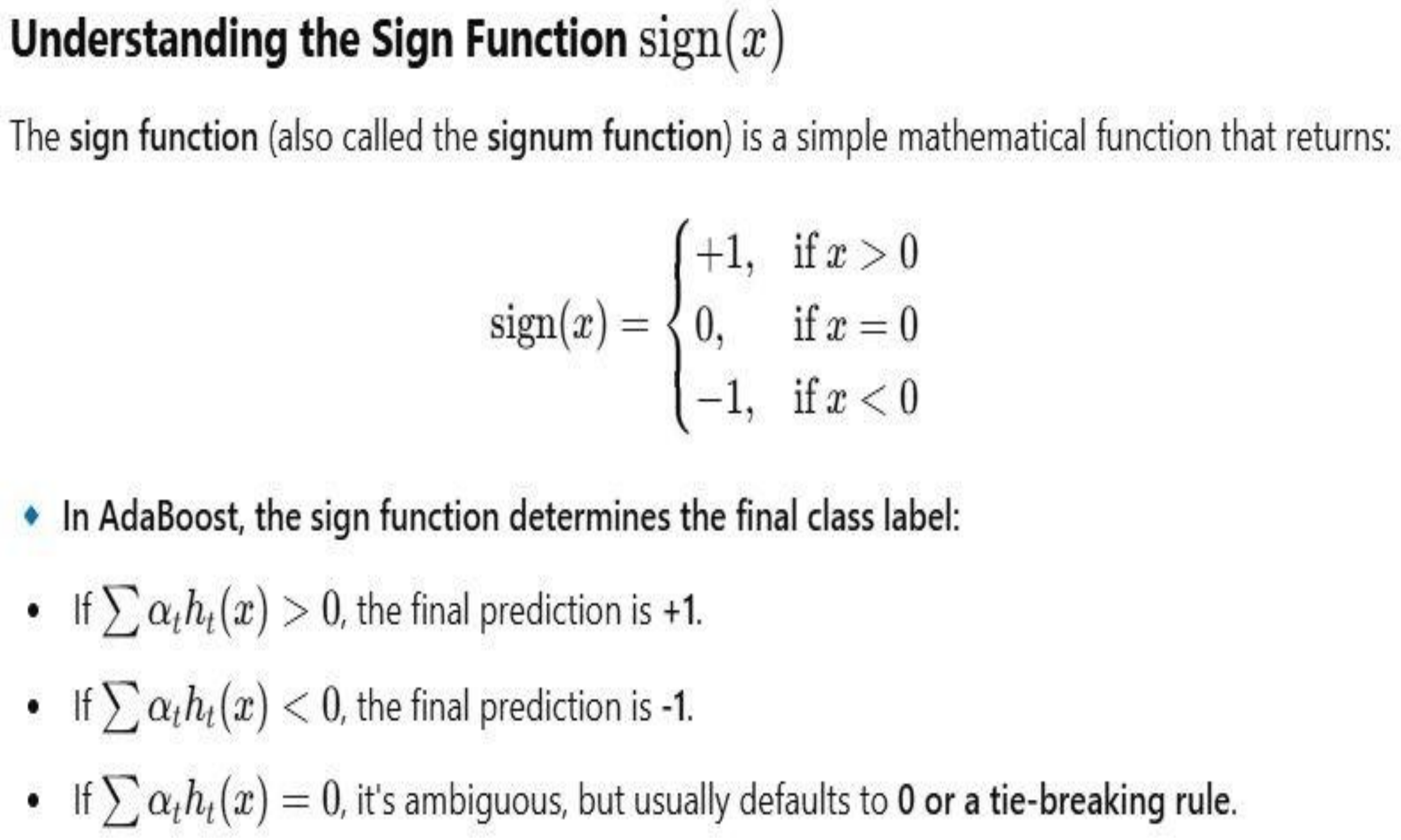


In **AdaBoost**, the final strong classifier is a **weighted combination** of all weak classifiers:

$$H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

where:

- $H(x)$ is the **final prediction** (strong classifier).
- α_t is the **weight** of each weak classifier $h_t(x)$.
- $h_t(x)$ is the **prediction** of the weak classifier (usually ± 1 for binary classification).
- T is the total number of weak classifiers.



Understanding the Sign Function $\operatorname{sign}(x)$

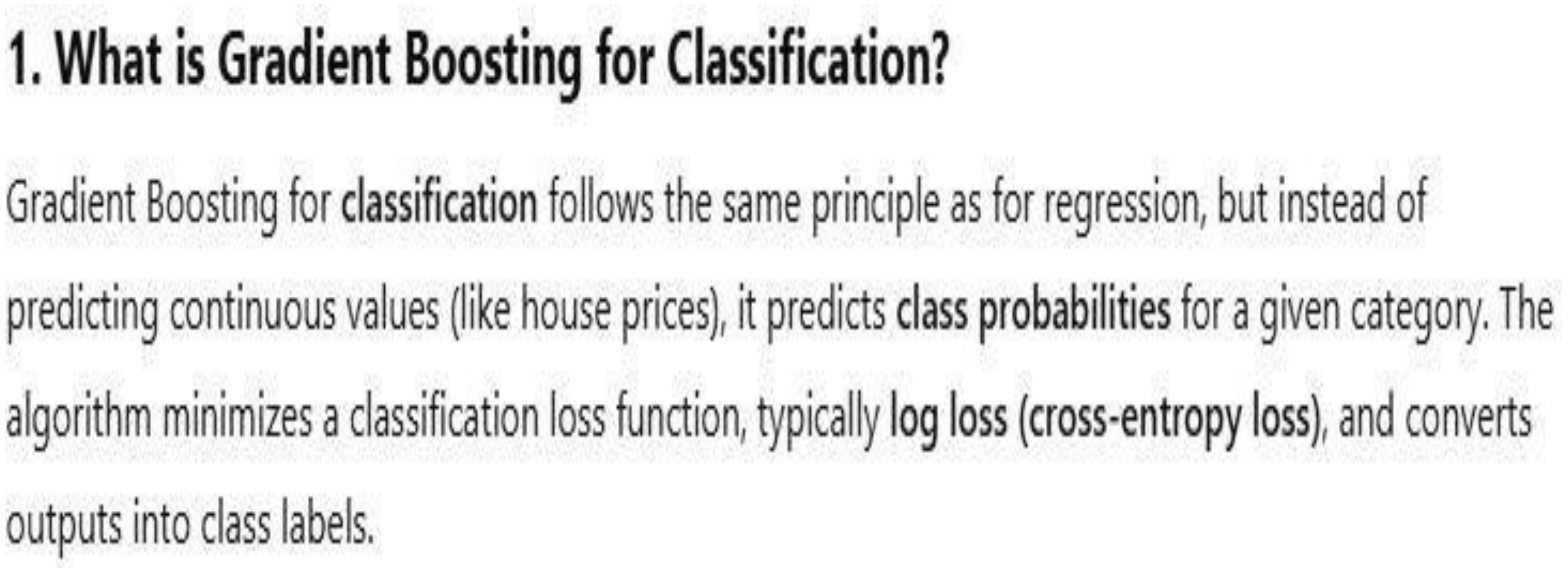
The **sign function** (also called the **signum function**) is a simple mathematical function that returns:

$$\operatorname{sign}(x) = \begin{cases} +1, & \text{if } x > 0 \\ 0, & \text{if } x = 0 \\ -1, & \text{if } x < 0 \end{cases}$$

🔹 **In AdaBoost, the sign function determines the final class label:**

- If $\sum \alpha_t h_t(x) > 0$, the final prediction is **+1**.
- If $\sum \alpha_t h_t(x) < 0$, the final prediction is **-1**.
- If $\sum \alpha_t h_t(x) = 0$, it's ambiguous, but usually defaults to **0 or a tie-breaking rule**.

Gradient Boosting: Classification



1. What is Gradient Boosting for Classification?

Gradient Boosting for **classification** follows the same principle as for regression, but instead of predicting continuous values (like house prices), it predicts **class probabilities** for a given category. The algorithm minimizes a classification loss function, typically **log loss (cross-entropy loss)**, and converts outputs into class labels.

2. How Gradient Boosting Works for Classification

Step 1: Initialize with a Weak Model

- The model starts with a weak baseline prediction, such as a **logarithmic transformation of class probabilities.**
- If we have **binary classification (0 or 1)**, we initialize:

$$F_0(x) = \log\left(\frac{p}{1-p}\right)$$

where p is the proportion of class 1 in the training dataset.

Step 2: Compute Pseudo-Residuals (Gradients)

- Compute the gradient of the loss function (log loss) for each sample.
- This gives us the **pseudo-residuals**, which indicate **how much each prediction needs to be corrected.**

$$r_i = y_i - p_i$$

where:

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

Step 3: Train a Weak Learner (Decision Tree)

- A **shallow decision tree** (depth=1) is trained to predict these pseudo-residuals.
- The tree finds patterns in misclassified points.

Step 4: Compute Step Size and Update Model

- Compute a step size γ that minimizes the loss function:

$$\gamma = \arg\min_{\gamma} \sum L(y_i, F_t(x_i) + \gamma h_t(x_i))$$

- Update the model with the new weak learner:

$$F_{t+1}(x) = F_t(x) + \eta \gamma h_t(x)$$

where η is the **learning rate.**

Step 5: Convert Predictions to Class Probabilities

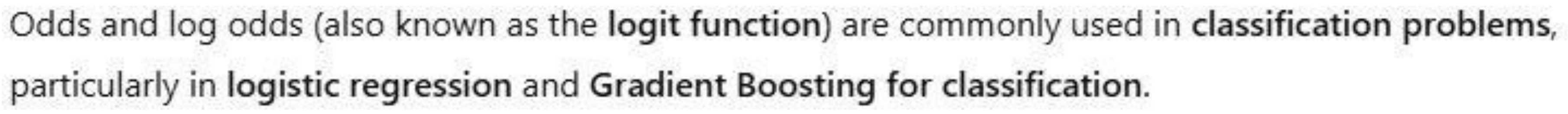
- Convert the final model's output to a probability using the **sigmoid function:**

$$P(y = 1|x) = \frac{1}{1 + e^{-F_T(x)}}$$

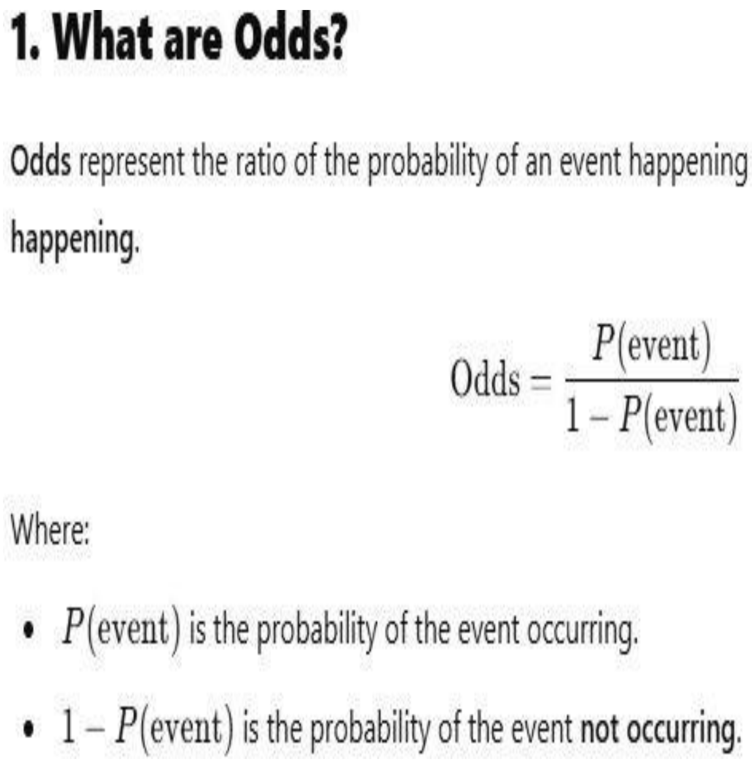
- Assign class labels based on a probability threshold (e.g., 0.5):

$$\hat{y} = \begin{cases} 1, & P(y = 1|x) > 0.5 \\ 0, & \text{otherwise} \end{cases}$$

Gradient Boosting: Odds and Log Odds



Odds and log odds (also known as the **logit function**) are commonly used in **classification problems**, particularly in **logistic regression** and **Gradient Boosting for classification**.



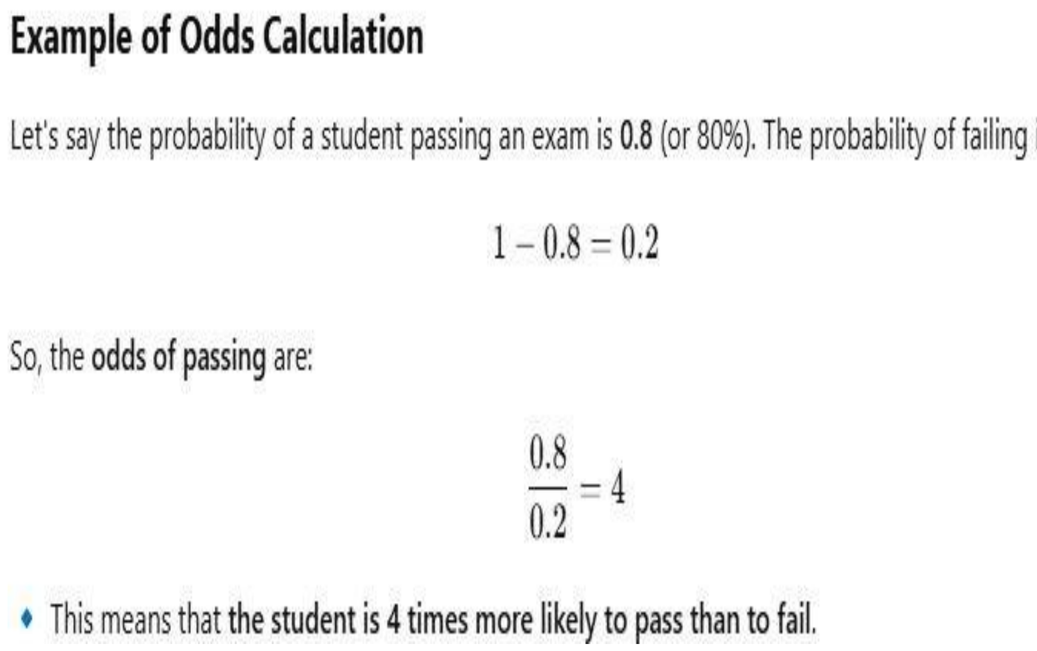
1. What are Odds?

Odds represent the ratio of the probability of an event happening to the probability of it **not** happening.

$$\text{Odds} = \frac{P(\text{event})}{1 - P(\text{event})}$$

Where:

- $P(\text{event})$ is the probability of the event occurring.
- $1 - P(\text{event})$ is the probability of the event **not occurring**.



Example of Odds Calculation

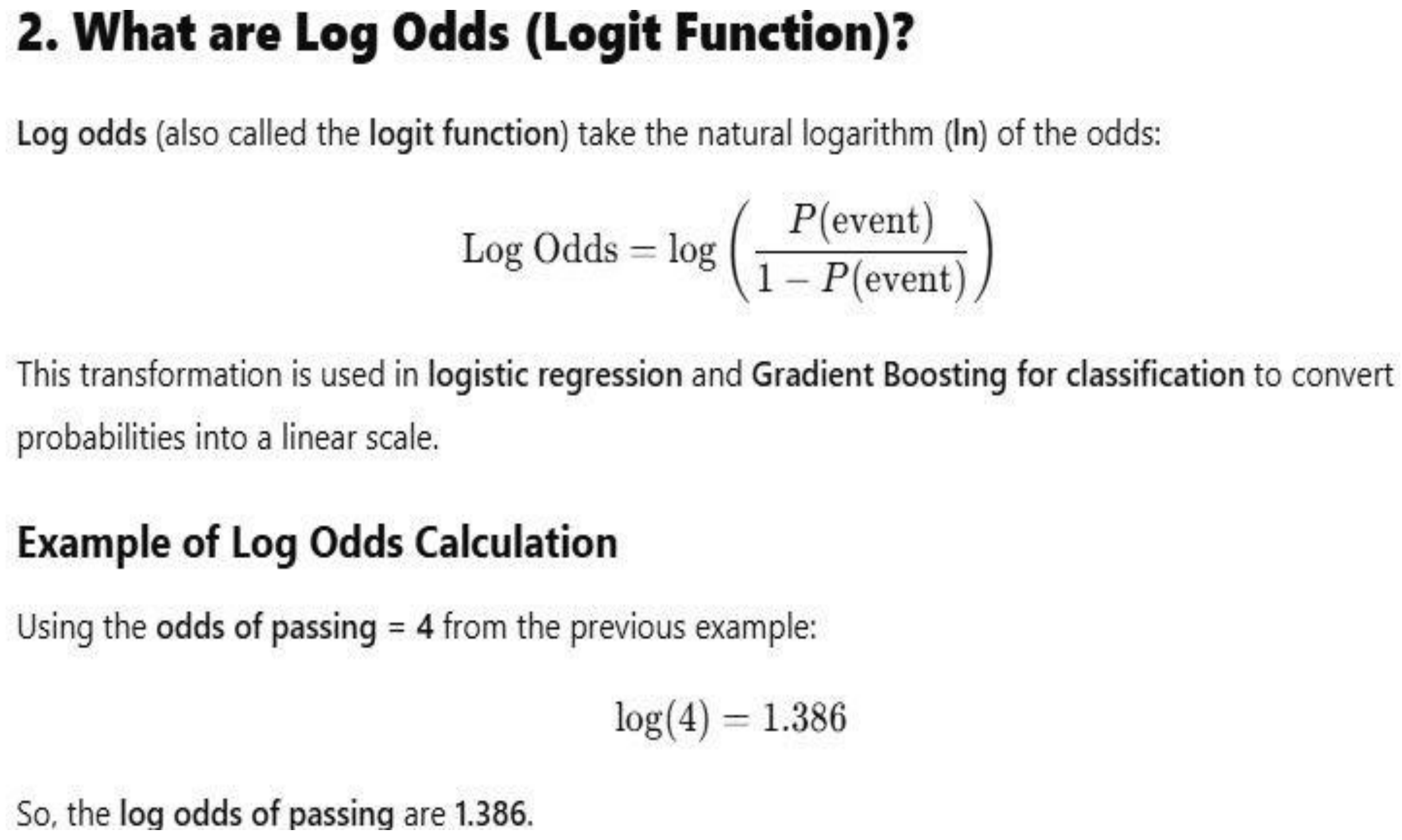
Let's say the probability of a student passing an exam is **0.8** (or 80%). The probability of failing

$$1 - 0.8 = 0.2$$

So, the **odds of passing** are:

$$\frac{0.8}{0.2} = 4$$

- This means that **the student is 4 times more likely to pass than to fail.**



2. What are Log Odds (Logit Function)?

Log odds (also called the **logit function**) take the natural logarithm (**ln**) of the odds:

$$\text{Log Odds} = \log\left(\frac{P(\text{event})}{1 - P(\text{event})}\right)$$

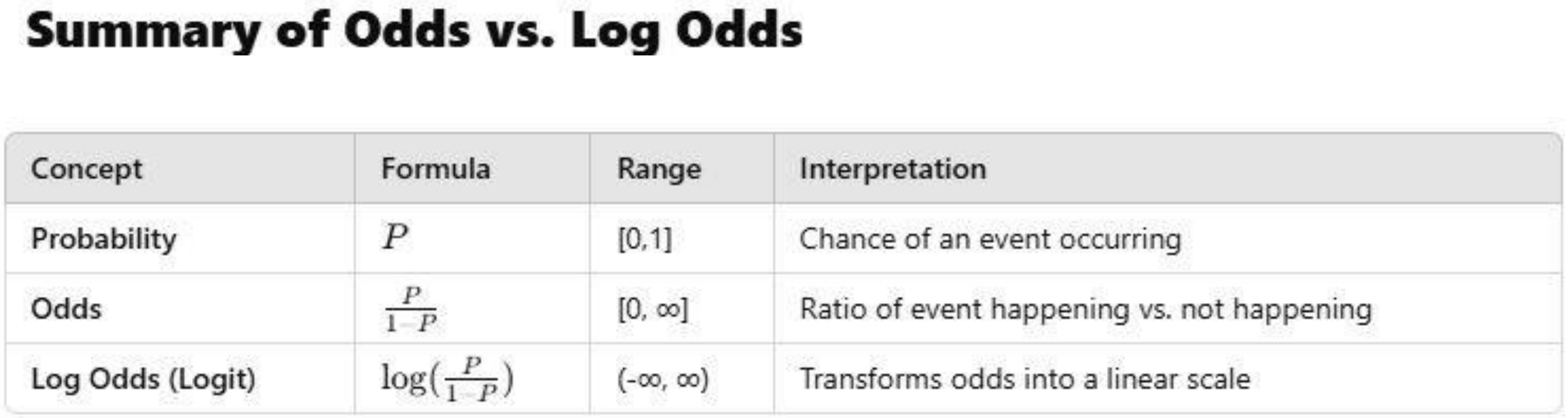
This transformation is used in **logistic regression** and **Gradient Boosting for classification** to convert probabilities into a linear scale.

Example of Log Odds Calculation

Using the **odds of passing** = **4** from the previous example:

$$\log(4) = 1.386$$

So, the **log odds of passing** are **1.386**.



Summary of Odds vs. Log Odds

Concept	Formula	Range	Interpretation
Probability	P	[0,1]	Chance of an event occurring
Odds	$\frac{P}{1-P}$	[0, ∞]	Ratio of event happening vs. not happening
Log Odds (Logit)	$\log(\frac{P}{1-P})$	(-∞, ∞)	Transforms odds into a linear scale

Thank You