Machine Learning

Video 96:

OOB Score | Out of Bag Evaluation in Random Forest | Machine Learning

Out-of-Bag (OOB) evaluation in machine learning is a validation technique used in bootstrap aggregating (bagging) methods like Random Forest. It estimates model performance without separate validation data by testing each tree on samples not used during training, reducing the need for cross-validation and providing an unbiased accuracy estimate.

The Out-of-Bag (OOB) error in a Random Forest model is calculated as:

$$\text{OOB Error} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{y}_i^{\text{OOB}})$$

Where:

- N = Total number of training samples
- y_i = True label of the i-th sample
- $\hat{y}_i^{\text{OOB}}$ = Predicted label from trees where i-th sample was **not** used for training
- $L(y, \hat{y})$ = Loss function (e.g., misclassification rate for classification or squared error for regression)

In **Random Forest**, approximately **37% of the training samples** are Out-of-Bag (OOB) for each tree.

Why 37%?

Each tree is trained on a **bootstrap sample**, where each sample has a **1−1n1 - \frac{1}{n}** chance of not being selected in one draw. For n total samples:

$$P(\text{sample not chosen in one draw}) = 1 - \frac{1}{n}$$

Since sampling is done **with replacement**, each sample has n opportunities to be picked. The probability that a given sample is **never picked** after n draws is:

$$P(\text{never chosen}) = \left(1 - \frac{1}{n}\right)^n$$

As n approaches infinity:

$$\lim_{n\to\infty}\left(1-\frac{1}{n}\right)^n \approx \frac{1}{e} \approx 0.368$$

So, around **36.8% (~37%)** of the rows are OOB for each tree.

Example:

Code link:

https://github.com/campusx-official/100-days-of-machine-learning/blob/main/day65-random-forest/oob-score-demo.ipynb

Video 97:

Feature Importance using Random Forest and Decision Trees | How is Feature Importance calculated

Feature importance refers to a technique used to determine how much each feature (input variable) contributes to a model's predictions. It helps in understanding which features are most influential in decision-making.

Why Feature Importance Matters?

- **Feature Selection** – Helps remove irrelevant features and improve model performance.
- **Interpretability** – Understand which features drive predictions.
- **Bias Detection** – Identify if a model is relying on unwanted biases (e.g., gender, race).

Example:

Code link:

https://github.com/campusx-official/100-days-of-machine-learning/blob/main/day65-random-forest/how-feature-importance-is-calculated.ipynb

Video 98:

How Adaboost Classifier works?

What are Weak learners?

In **AdaBoost (Adaptive Boosting)**, a **weak learner** is a model that performs slightly better than random guessing (i.e., has an accuracy just above 50% for binary classification).

Why Weak Learners?

- **Prevent Overfitting** – Using simple models avoids excessive complexity.
- **Boosting Strengthens Them** – Multiple weak learners together form a strong classifier.

What are Decision stumps?

A Decision Stump is a weak learner used in AdaBoost. It is a simplified Decision Tree with only one split (one level deep).

Video 99:

AdaBoost - A Step-by-Step Explanation:

1 Initialize Weights for All Samples

- Start with **equal weights** for all training samples:

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

  where N is the total number of training examples.
- These weights help determine how much importance each sample has in training.

---

2 Train a Weak Learner (e.g., Decision Stump)

- Fit a weak model (like a Decision Stump) to the **weighted dataset**.
- The weak learner makes predictions on the dataset.

3 Compute Weighted Error

- Calculate the weighted error of the weak learner:

$$\text{Error} = \sum_{i=1}^{N} w_i \cdot I(y_i \neq \hat{y}_i)$$

  where:
  - $I(y_i \neq \hat{y}_i)$ is **1** if the prediction is wrong, **0** otherwise.
  - w_i is the weight of sample i.
- If the weak learner performs well (low error), it gets a **higher weight in final prediction**.

4 Compute Alpha (Model Weight)

- Compute the model's **weight (α)** based on its error:

$$\alpha = \frac{1}{2} \ln \left(\frac{1 - \text{Error}}{\text{Error}} \right)$$

- If **error is low**, α is high (the weak model is more important).
- If **error is high**, α is low (the weak model is less important).

---

5 Update Sample Weights

- Increase weights for **misclassified samples**, so the next weak learner focuses on them:

$$w_i^{\text{new}} = w_i^{\text{old}} \cdot e^{\alpha \cdot I(y_i \neq \hat{y}_i)}$$

- Normalize weights so they sum to **1**:

$$w_i = \frac{w_i}{\sum w_i}$$

- Now, misclassified points have **higher importance** in the next iteration.

6 Repeat for Multiple Iterations

- Train the next weak learner on the **updated sample weights**.
- Repeat steps **2 to 5** for multiple iterations.
- Each new weak learner corrects errors from previous learners.

---

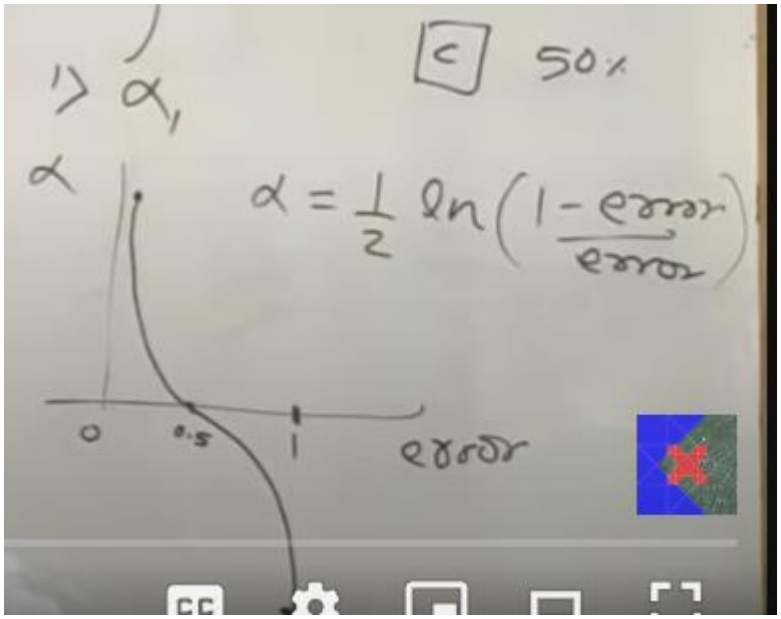
7 Final Prediction (Strong Classifier)

- Combine all weak learners using their weights (α) for a final prediction:

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

  - $H(x)$ is the final classifier.
  - $h_t(x)$ are the weak classifiers.
  - α_t are their weights.

Clearly,



Video 99:

AdaBoost Algorithm | Code from Scratch

Example:

Code link**:**
https://colab.research.google.com/drive/15sn5CZtyr67NjqVRou84p47Fx7zvk6eD?usp=sharing