Decision Trees in Machine Learning 🌳

1.What is a Decision Tree?

A **decision tree** is a supervised learning algorithm used for **classification** and **regression** tasks. It splits the data into branches based on feature conditions, forming a tree-like structure.

- **Root Node** → The first decision point.
- **Internal Nodes** → Intermediate decisions based on conditions.
- **Leaf Nodes** → Final output (class label or numerical value).

2.How Does a Decision Tree Work?

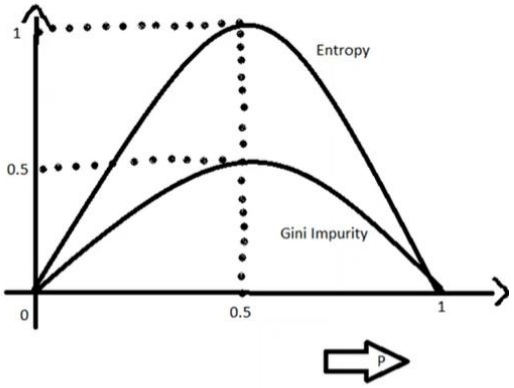
1. Start at the **root node**.
2. Split the data at each node using a **feature** that provides the best information gain.
3. Continue splitting until:
    - A stopping condition is met (e.g., max depth reached).
    - All data points in a node belong to one class.
4. Assign labels to the leaf nodes.

The algorithm chooses splits based on:

- **Gini Impurity** → Measures how often a randomly chosen element would be incorrectly labeled.
- **Entropy (Information Gain)** → Measures the uncertainty in a dataset.

Entropy vs Gini

Can be computed by summing the probability of an item with label i being chosen (Pi), times the probability of a mistake (1—Pi) in categorizing that item



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

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

3.Advantages & Disadvantages

✅ **Advantages:**

✔ Easy to understand & interpret.

✔ Works with both numerical & categorical data.

✔ Handles non-linear relationships well.

❌ **Disadvantages:**

❌ Can **overfit** (perform poorly on new data).

❌ Sensitive to **noisy** data.

❌ Decision trees alone may not be the most accurate model (but can be improved using **Random Forests**).

4.Python Code Example

Here's how you can build a decision tree using sklearn:

```
from sklearn.datasets import load_iris

from sklearn.tree import DecisionTreeClassifier, plot_tree

import matplotlib.pyplot as plt


# Load dataset

iris = load_iris()
```

```
X, y = iris.data, iris.target


# Train Decision Tree model
model = DecisionTreeClassifier(criterion="gini", max_depth=3, random_state=42)
model.fit(X, y)


# Visualize the Decision Tree
plt.figure(figsize=(12, 6))
plot_tree(model, feature_names=iris.feature_names, class_names=iris.target_names, filled=True)
plt.show()
```

5.Variations & Enhancements

- **Random Forest** 🌲 → Uses multiple decision trees to improve accuracy.
- **Gradient Boosting** 🚀 → Sequentially improves weak trees.
- **XGBoost** 🔥 → Optimized version of gradient boosting.

6.Overfitting vs. Underfitting

In machine learning, overfitting and underfitting are two common problems that affect model performance. Let's break them down:

1.Overfitting

🔴 **Definition:**

- **The model learns too much from the training data, including noise and irrelevant patterns.**
- **It performs extremely well on training data but poorly on new (test) data.**

🔴 **Causes:**

✅ **Model is too complex (e.g., deep decision trees).**

✅ **Not enough training data.**

✅ **Model is trained for too many epochs (in neural networks).**

✅ **Features include too much noise or irrelevant information.**

🔴 **Symptoms:**

- **High accuracy on training data but low accuracy on test data.**
- **Model predicts the training data perfectly but fails on unseen data.**

🔴 **Solution to Overfitting:**

✓ **Pruning the decision tree (for Decision Trees).**

✓ **Reduce model complexity (e.g., limit max_depth in Decision Trees).**

✓ **Use more training data to generalize better.**

✓ **Regularization techniques (L1, L2, Dropout in neural networks).**

✓ **Cross-validation to check model performance.**

2.Underfitting

🟢 **Definition:**

- **The model is too simple and cannot learn the patterns from the training data.**
- **It performs poorly on both training and test data.**

🟢 **Causes:**

✅ **Model is too simple (e.g., very shallow decision trees).**

✅ **Not enough training time (for deep learning models).**

✅ **Too few features → Important information is missing.**

✅ **Wrong algorithm choice for a complex problem.**

🟢 **Symptoms:**

- **Low accuracy on both training and test data.**
- **Model fails to capture underlying trends in the data.**

🟢 **Solution to Underfitting:**

✓ Increase model complexity (e.g., allow deeper trees).
✓ Train for longer (for neural networks).
✓ Add more relevant features.
✓ Use a more powerful model (e.g., switch from linear regression to polynomial regression).

Feature	Overfitting 🔴	Underfitting ⚠️
Training Accuracy	🔥 Very High	❄️ Low
Test Accuracy	❌ Very Low	❌ Low
Model Complexity	🚀 Too Complex	🏚️ Too Simple
Generalization	❌ Poor	❌ Poor
Solution	Regularization, Pruning, More Data	Increase Complexity, Add Features

---

Overfitting = Too much learning (memorization instead of generalization).
Underfitting = Too little learning (cannot capture patterns).
A good model balances both (generalizes well to unseen data).

Eg:

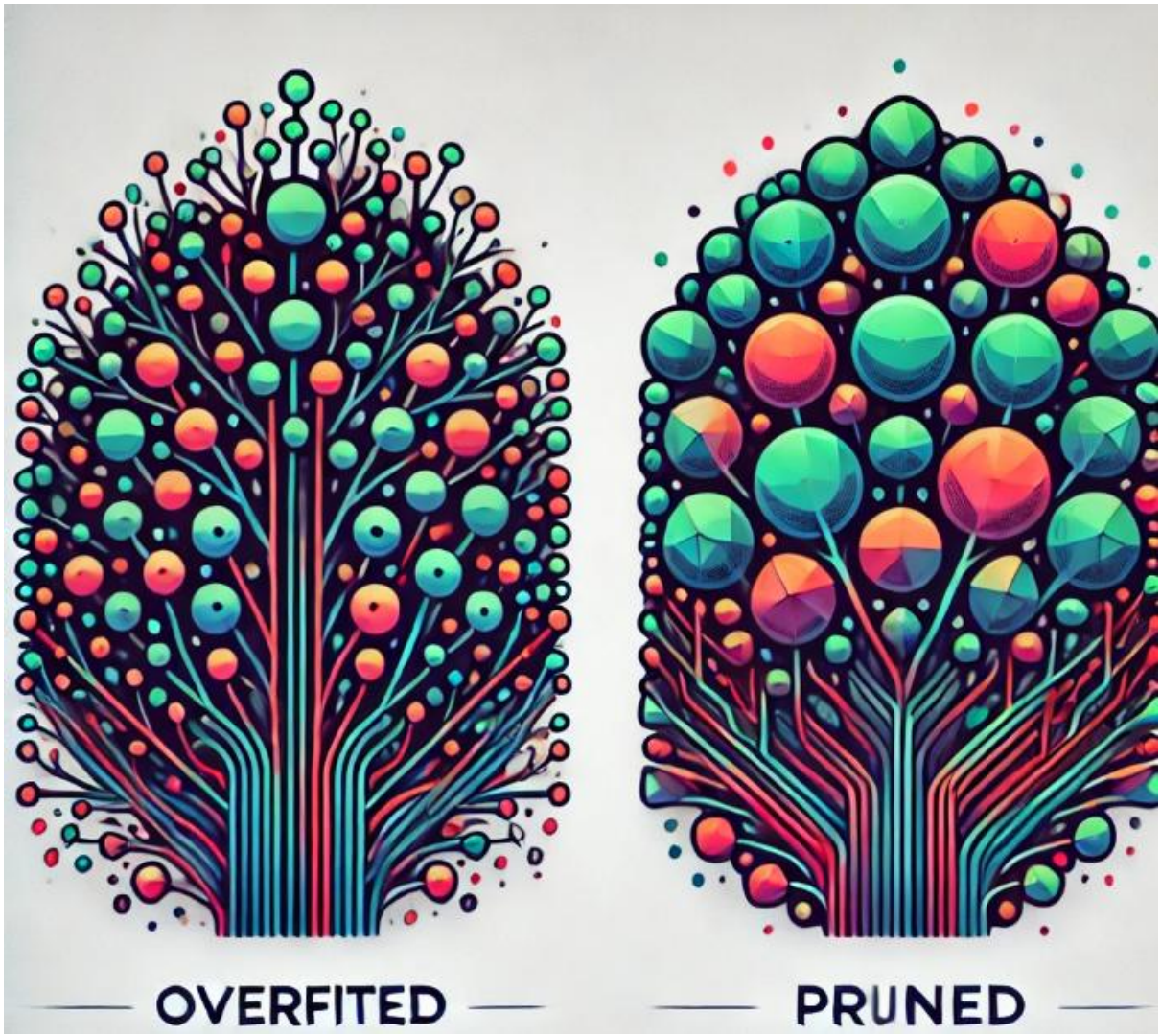
```
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy_score

from sklearn.model_selection import train_test_split


# Sample dataset

X = [[5, 120, 70], [2, 140, 80], [8, 130, 75], [6, 125, 72], [3, 145, 85]]

y = [0, 1, 0, 0, 1]  # Labels
```

```
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Overfitting Example (Very deep tree)
overfit_model = DecisionTreeClassifier(max_depth=10)
overfit_model.fit(X_train, y_train)
print("Overfitting Accuracy:", accuracy_score(y_test, overfit_model.predict(X_test)))

# Underfitting Example (Very shallow tree)
underfit_model = DecisionTreeClassifier(max_depth=1)
underfit_model.fit(X_train, y_train)
print("Underfitting Accuracy:", accuracy_score(y_test, underfit_model.predict(X_test)))
```

6. 🌳 Pruning in Decision Trees

What is Pruning?

Pruning is a technique used to reduce the size of a decision tree by removing branches that do not provide significant predictive power. This helps in reducing overfitting and improving the model's generalization to new data.


OVERFITED
PRUNED