Support Vector Machines, Decision Trees, and Random Forests

Artificial Intelligence (Machine Learning & Deep Learning)

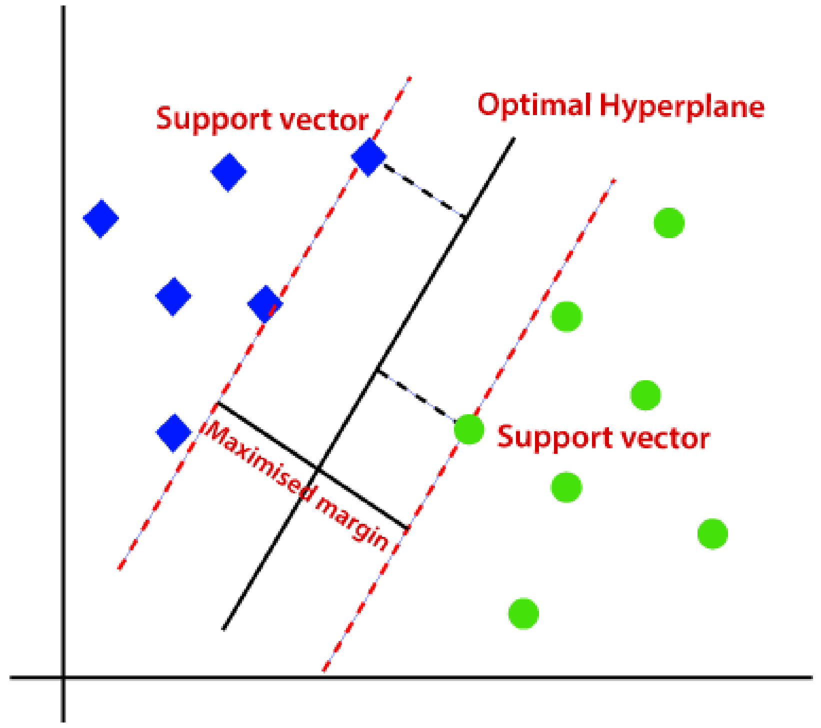
Support Vector Machines (SVM)

Background

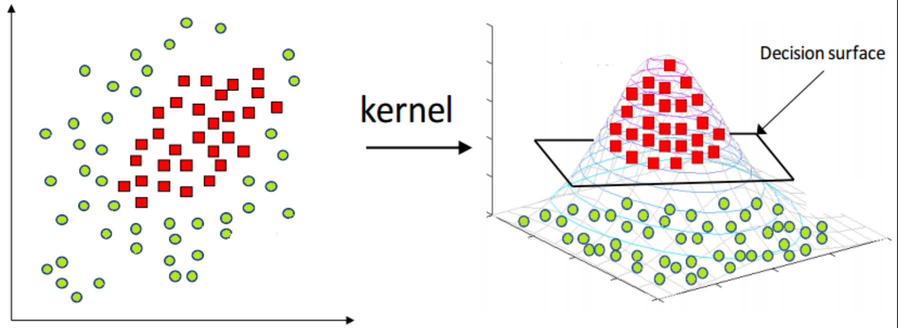
- Support Vector Machines (SVM) are powerful supervised learning algorithms used for both classification and regression. They work by finding an optimal decision boundary (hyperplane) that separates different classes with the maximum margin.

➢ Brief History:

- Introduced by Vladimir Vapnik and Alexey Chervonenkis in 1963 at the Institute of Control Sciences, USSR.
- Further refined in the 1990s, leading to the soft-margin SVM and the kernel trick for handling non-linear data.

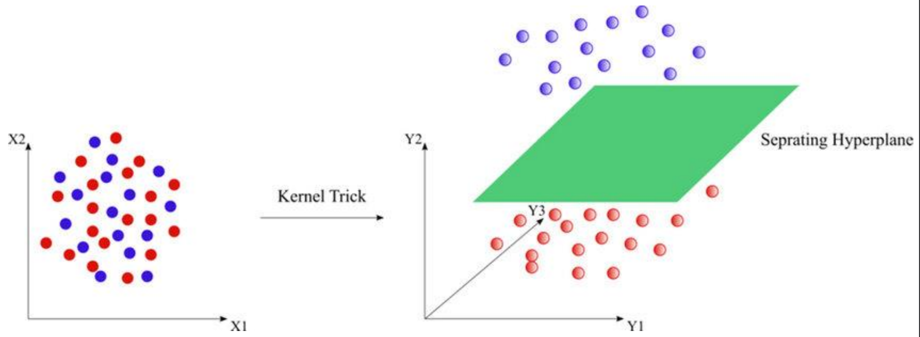


How SVM Works



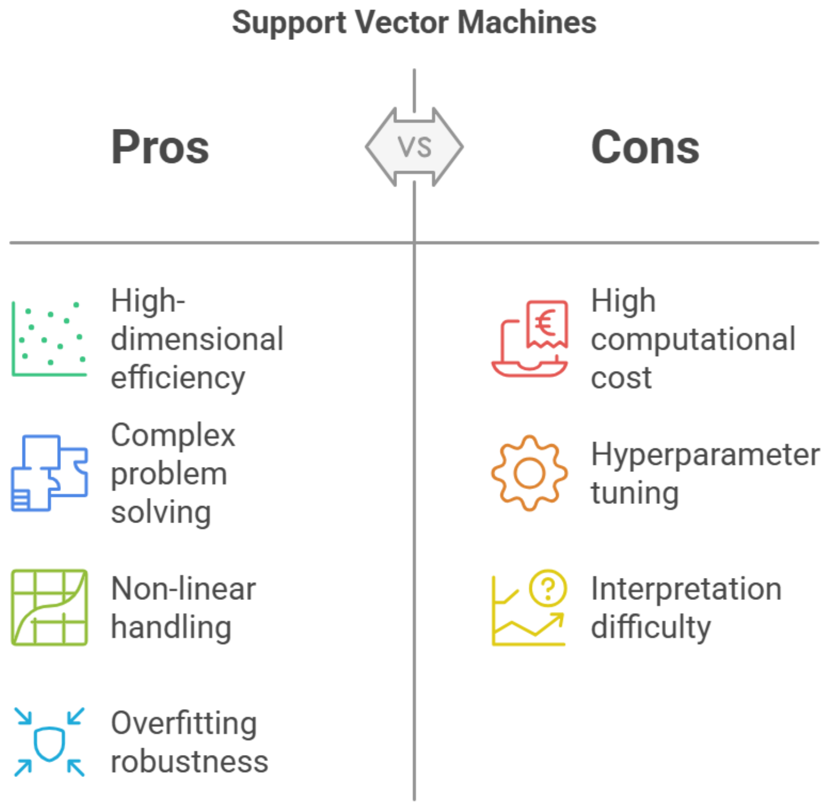
- **Finding the Optimal Hyperplane:**
  - SVM tries to maximize the margin between two classes.
  - The hyperplane is a decision boundary that best separates the data points.

- **Key Concepts:**
  - **Linear Separation**: In a 2D space, a hyperplane is a line that separates two classes. In higher dimensions, it's a plane or a surface.
  - **Support Vectors:** Data points closest to the hyperplane, influencing its position.
  - **Margin:** Distance between the hyperplane and the nearest data points. A larger margin leads to better generalization.
  - **Kernel Trick**: Enables SVM to handle non-linearly separable data by transforming it into a higher-dimensional space.



Strengths & Limitations

- **Strengths**
  - Effective in high-dimensional spaces.
  - Works well for complex classification problems.
  - Can handle non-linear relationships using kernel functions.
  - Robust to overfitting, especially when the number of features is large.
- **Limitations:**
  - Computationally expensive for large datasets.
  - Requires careful hyperparameter tuning (kernel, C, gamma).
  - Results may be difficult to interpret compared to decision trees.



Support Vector Machines

When to Use SVM?

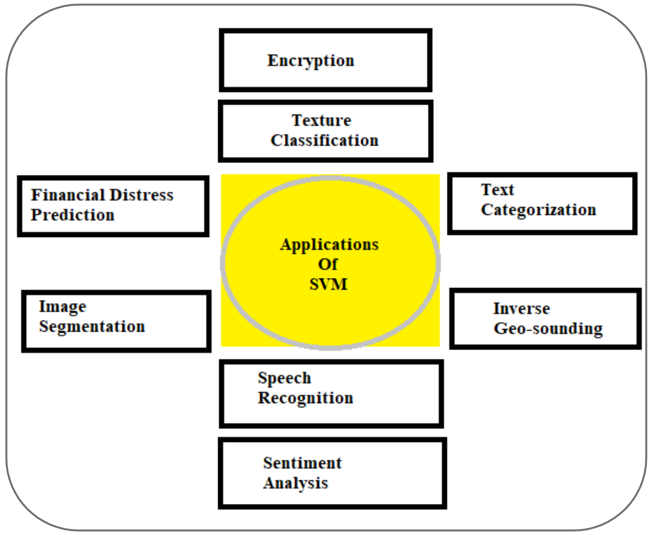
- When high accuracy is required, and the dataset is small to medium-sized.
- When the data has clear margins of separation.
- Suitable for binary classification tasks.
- Works well when the number of features is large compared to the number of samples.

Example Use Case

- **Spam detection** – Classifying emails as spam or not spam.
- **Image classification** – Recognizing objects in images.
- **Medical diagnosis** – Identifying diseases based on patient data.

Python Implementation

```
from sklearn.svm import SVC
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load dataset
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Train SVM model
model = SVC(kernel='rbf', C=1.0, gamma='scale')
model.fit(X_train, y_train)

# Predictions and accuracy
predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test, predictions))
```



SVM - Hypothesis, Cost Function, and Optimization

1. **Hypothesis (Decision Boundary)**
   - SVM finds the optimal hyperplane that maximizes the margin between two classes.
   - For linear SVM, the decision function is:

$$h_\theta(x) = \theta^T x + b$$

   - The classification rule:

$$h_\theta(x) = \begin{cases} 1, & \theta^T \geq 0 \\ 0 & otherwise \end{cases}$$

2. **Cost Function (Hinge Loss)**
   - To enforce a large margin, we use the hinge loss function:

$$J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m}\left(\max(0, 1 - y^{(i)}(\theta^T x^{(i)} + b))\right) + \frac{\lambda}{2}\|\theta\|^2$$

3. **Optimization Algorithm (Gradient Descent for SVM)**
   - To minimize the cost function, we update θ and b using Gradient Descent.
   - Gradient updates:

$$\theta := \theta - \alpha\left(\lambda\theta - \sum_{i=1}^{m} 1(y^{(i)}(\theta^T x^{(i)}) < 1)y^{(i)}x^{(i)}\right)$$

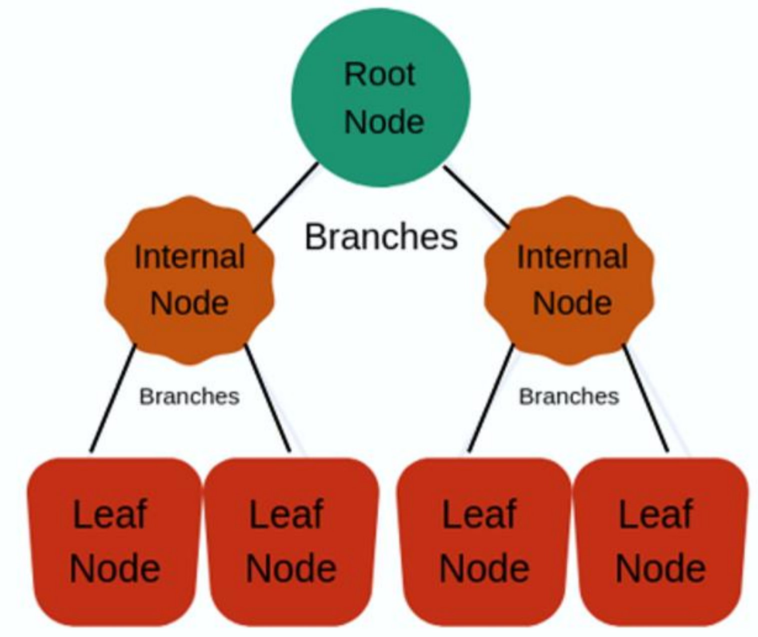
Key differences from Linear/Logistic Regression:
- SVM optimizes for maximum margin rather than minimizing error
- Uses hinge loss instead of squared/log loss
- Incorporates kernel functions for non-linear classification

Decision Trees

Decision Trees

Background

- Decision Trees are supervised learning algorithms used for classification and regression.
- They work by recursively splitting data into subsets based on feature values, forming a tree-like structure.



➢ **Key Components:**

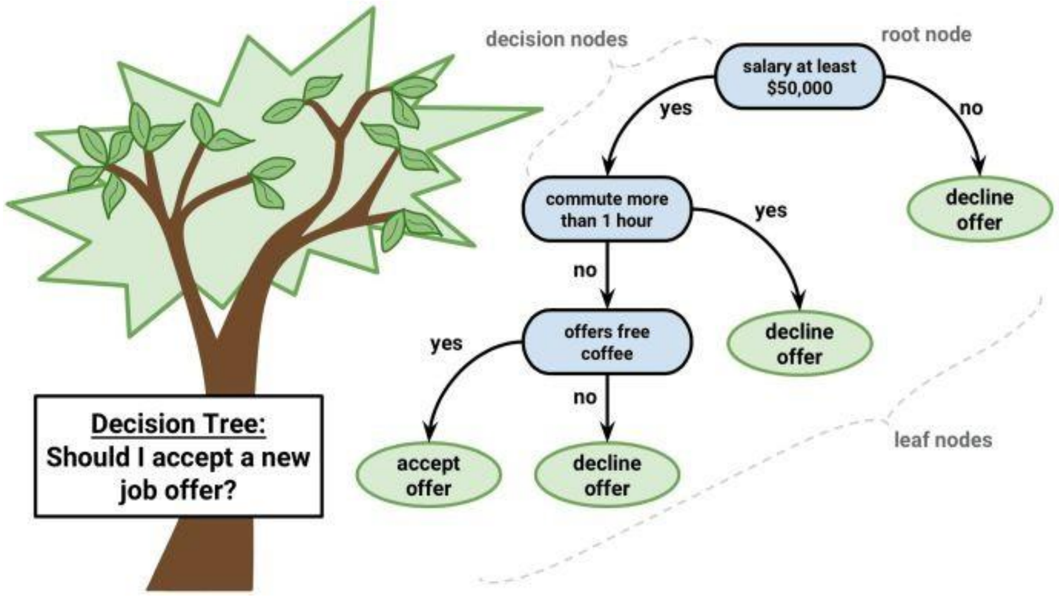
- **Root Node**: Represents the entire dataset.
- **Internal Nodes:** Represent decisions based on feature values.
- **Branches:** Represent outcomes of decisions.
- **Leaf Nodes:** Represent final predictions (class labels or values).

➢ **Brief History:**

- The concept of Decision Trees emerged in the 1960s through research in pattern recognition and decision theory.
- Ross Quinlan formalized the algorithm in 1986 with ID3 (Iterative Dichotomiser 3) at the University of Sydney.
- Later advancements introduced C4.5 (1993) and CART (Classification and Regression Trees, 1984), which are widely used today.

How Decision Trees Work

- **Tree Construction:**
  - The tree starts with a root node containing the full dataset.
  - At each step, the algorithm selects the best feature to split the data, based on criteria such as:
    - **Information Gain** (used in ID3, C4.5)
    - **Gini Impurity** (used in CART)
  - **Splitting Criteria**:
    - The dataset is split into branches based on feature values.
    - The process continues until a stopping condition is met (e.g., maximum depth, minimum samples per node.
  - **Making Predictions:**
    - To classify a new data point, start at the root node and follow the branches based on feature values until reaching a leaf node (final decision).



Strengths & Limitations

➢ **Strengths**

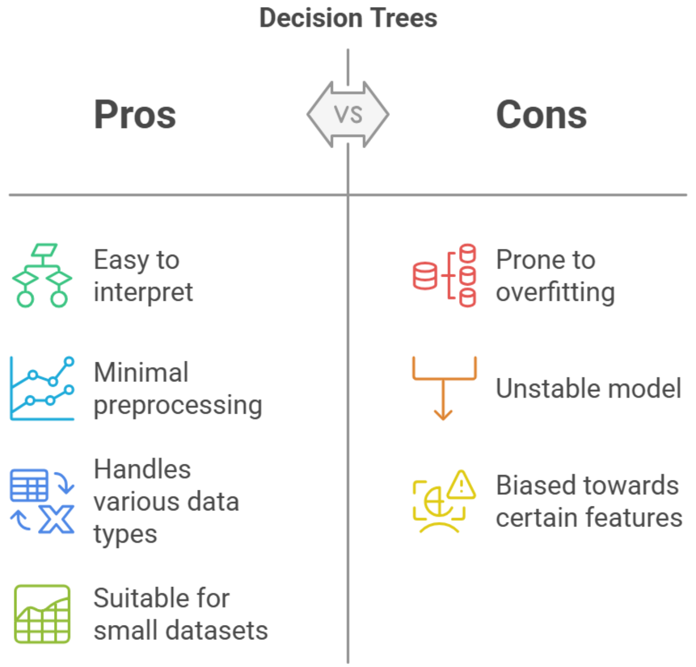
- Easy to interpret and visualize.
- Minimal data preprocessing required – No need for feature scaling.
- Handles both numerical and categorical data.
- Works well on small medium – sized datasets.

➢ **Limitations**

- Prone to overfitting – Deep trees may memorize the training data..
- Unstable model – Small changes in data can lead to different trees.
- Biased towards features with more categories.

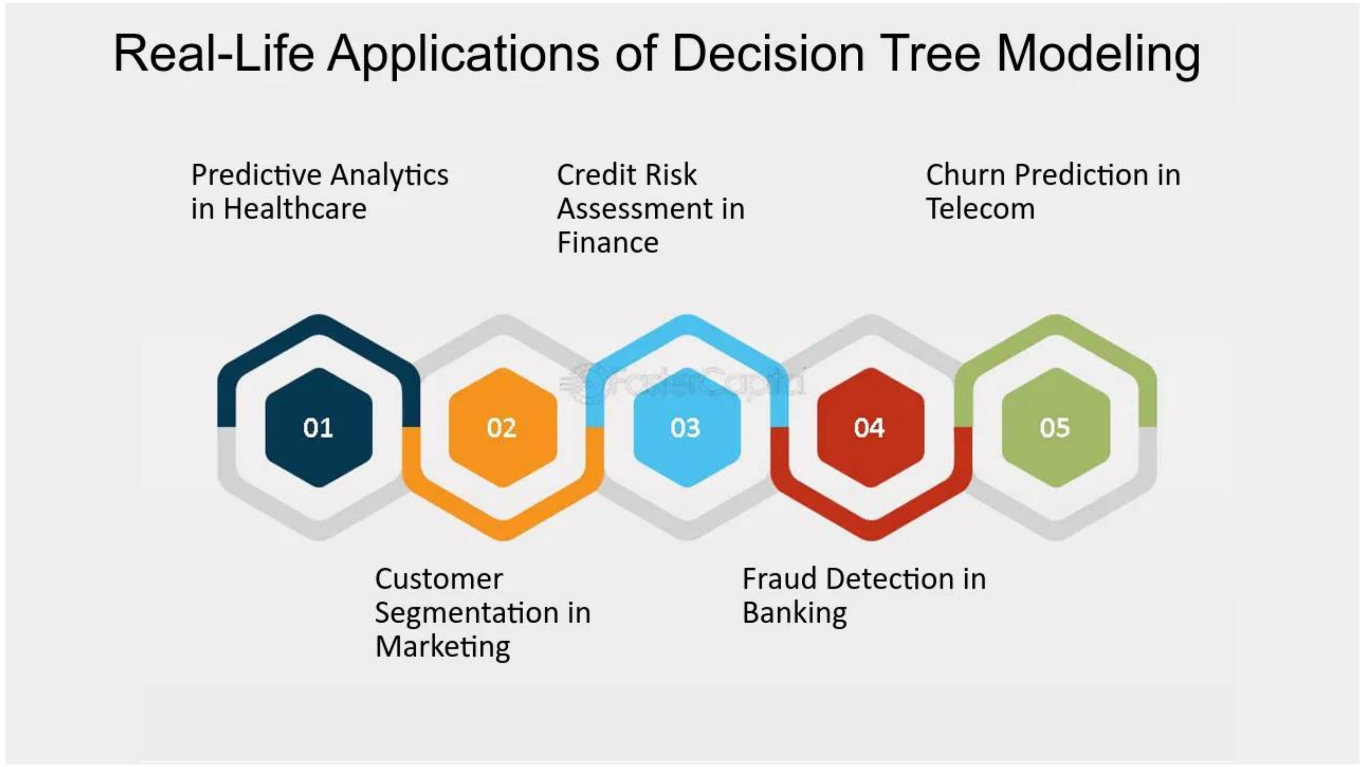
Python Implementation

```
from sklearn.tree import DecisionTreeClassifier

# Train Decision Tree model
model = DecisionTreeClassifier(max_depth=3,
random_state=42)
model.fit(X_train, y_train)

# Predictions and accuracy
predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test,
predictions))
```



When to Use Decision Trees?

- When interpretability is important – easy to explain and understand.
- When working with mixed data types – can handle both categorical and numerical features.
- As a baseline model – a quick, effective starting point before using complex models.

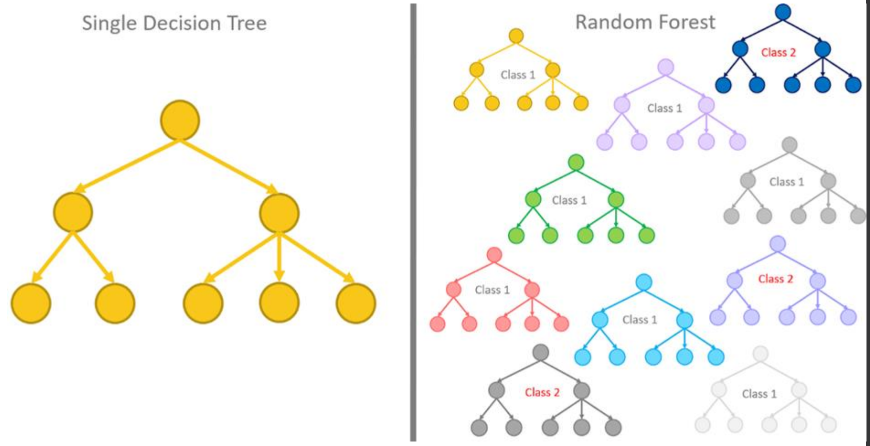

Real-Life Applications of Decision Tree Modeling
Predictive Analytics in Healthcare
Credit Risk Assessment in Finance
Churn Prediction in Telecom
01
02
03
04
05
Customer Segmentation in Marketing
Fraud Detection in Banking

Random Forests

Random Forests

Background

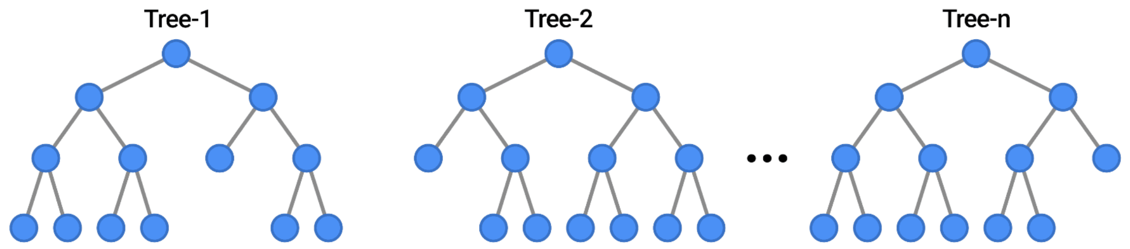
- Random Forest is an ensemble learning method that improves the performance of Decision Trees by creating multiple trees and aggregating their predictions.
- Introduced by Leo Breiman and Adele Cutler (2001, UC Berkeley) to reduce overfitting and enhance accuracy in classification and regression tasks.



➢ Key Concept:

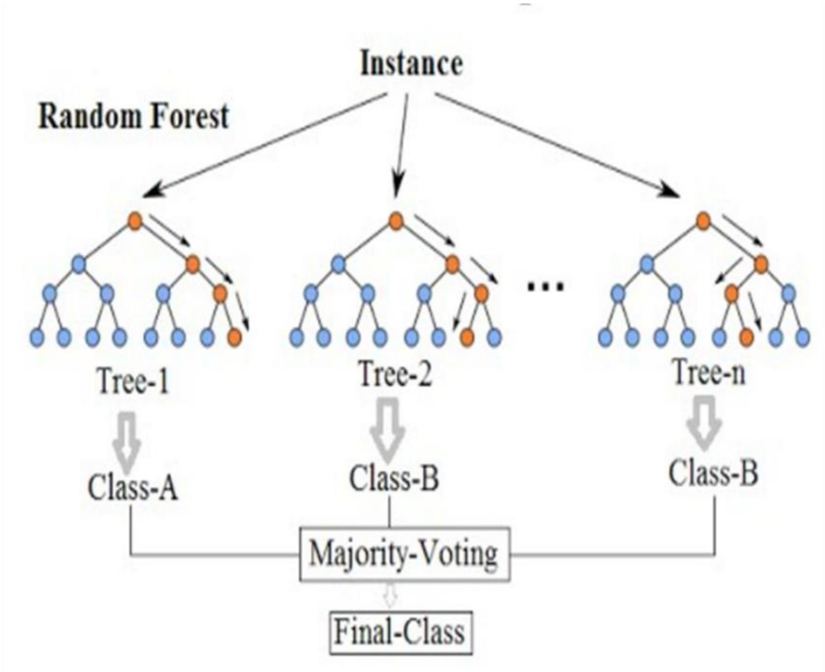
- Instead of relying on a **single Decision Tree**, Random Forest builds **multiple trees** and combines their outputs for a more **robust** and **generalized** prediction.

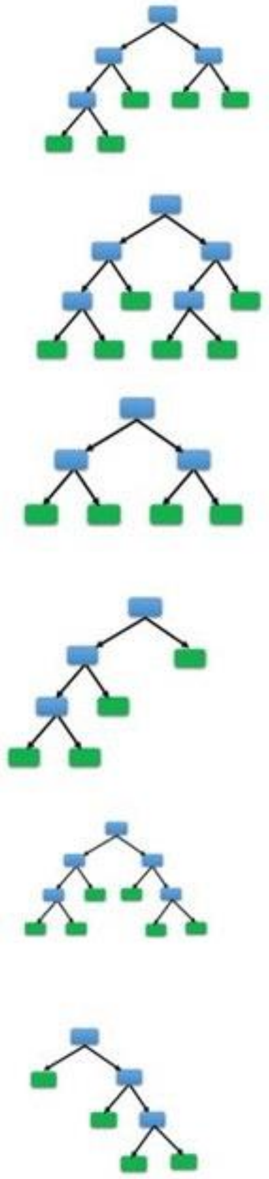
EXAMPLES



How Random Forests Works

1. **Bootstrap Aggregating (Bagging):**
   - Multiple Decision Trees are trained on different random subsets of the dataset (with replacement).
2. **Feature Randomness:**
   - At each split, a random subset of features is considered instead of using all features.
   - This introduces diversity, preventing trees from learning the same patterns.
3. **Voting & Averaging:**
   - **For classification:** The majority class predicted by the trees is chosen (majority voting).
   - **For regression:** The average of all tree predictions is taken (mean aggregation).



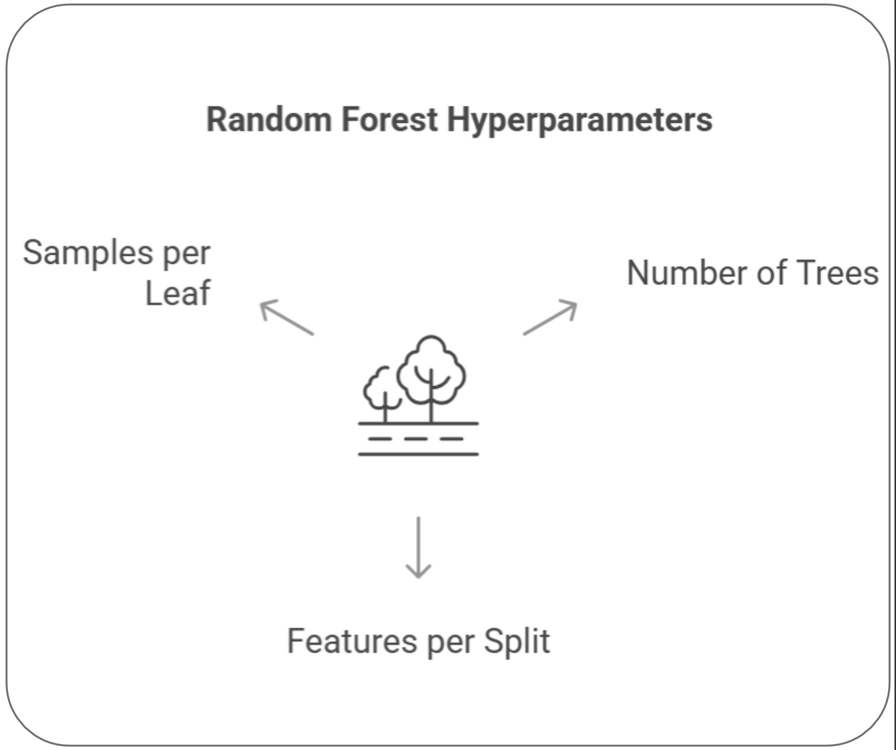


Random Forest in Action!!!

Hyperparameters & Tuning in Random Forest

Three key parameters influence Random Forest performance:

- **Number of Trees (n_estimators):**
  - Defines the number of trees in the forest.
  - 500 trees is often a good starting point in practice.
- **Number of Features Sampled per Split (max_features):**
  - Controls how many features are considered at each node split.
  - Sampling **2-5** features per split is usually effective.
- **Minimum Samples per Leaf (min_samples_leaf):**
  - Unlike single Decision Trees, each leaf node can have fewer observations to reduce bias.
  - Helps balance depth and generalization.



Strengths & Limitations

- **Strengths**
  - Reduces overfitting compared to individual decision trees.
  - Handles high-dimensional data well.
  - Robust to outliers and noise.
  - Provides feature importance scores.
- **Limitations**
  - **Computationally expensive** – Training hundreds of trees requires more memory & processing power.
  - **Less interpretable than a single tree** – Harder to explain predictions.
  - **Requires careful tuning** – Needs adjustments for the number of trees, depth, and sampled features.

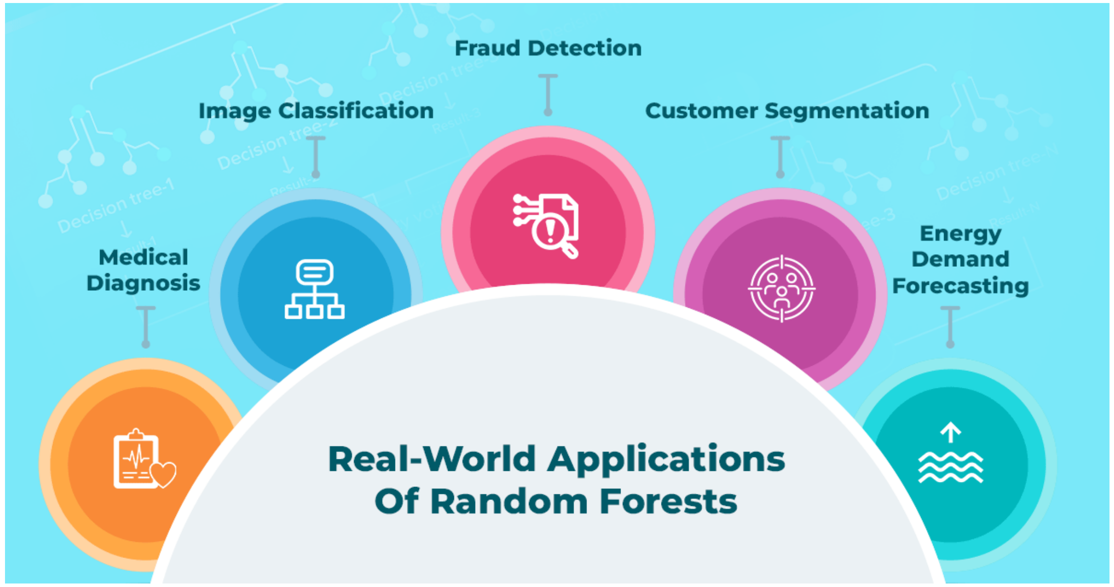
Python Implementation

```
from sklearn.ensemble import
RandomForestClassifier

# Train Random Forest model
model =
RandomForestClassifier(n_estimators=100,
max_depth=5, random_state=42)
model.fit(X_train, y_train)

# Predictions and accuracy
predictions = model.predict(X_test)
print("Accuracy:", accuracy_score(y_test,
predictions))
```

When to Use Random Forest?

- When accuracy is a priority over interpretability.
- When working with large, high-dimensional datasets.
- When reducing overfitting from individual Decision Trees.

- **Example Use Case**
  - **House Price Prediction** – Estimating property values based on location, size, and features.
  - **Fraud Detection** – Identifying suspicious financial transactions.
  - **Medical Diagnosis** – Classifying diseases based on patient records.



Comparison of SVM, Decision Trees, and Random Forests

Model	Strengths	Limitations	Best Use Cases
SVM	Works well with high-dimensional data, handles non-linear data	Computationally expensive, needs careful tuning	Text classification, bioinformatics
Decision Tree	Easy to interpret, no need for feature scaling	Prone to overfitting, sensitive to small data changes	Small datasets, when interpretability is crucial
Random Forest	Reduces overfitting, works well on large datasets	Computationally expensive, less interpretable	High-accuracy tasks, large and complex datasets

Thank You