Machine Learning Algorithms And Models

🔷 1. Decision Tree

Definition: A Decision Tree splits data into smaller subsets based on feature conditions, forming a tree structure. It uses impurity measures (like **Entropy** or **Gini**) to decide the best splits.

📌 Key Terms:

Root Node: First node; contains the full dataset.

Internal Node: Splits based on a feature.

Leaf Node: Final output (class or value).

🧮 Equations:

Entropy:

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

Gini Index:

$Gini = 1 - \sum_{i=1}^{c} p_i^2$

🛠️ Steps:

Choose the best feature (using Gini/Entropy).

Split dataset.

Repeat until stopping criteria (max depth, min samples).

🔢 Example (Python):

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris

X, y = load_iris(return_X_y=True)
model = DecisionTreeClassifier(max_depth=3)
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

Easy to interpret & visualize.

Handles numeric & categorical data.

❌ Disadvantages:

Prone to overfitting.

Sensitive to small changes in data.

🎯 Use Cases:

Fraud detection, medical diagnosis.

🔷 2. Random Forest

Definition: An **ensemble method** that builds multiple Decision Trees on random subsets of data & features, then averages predictions to improve accuracy.

🧮 Equation:

For classification:

$\acute{y}\wedge = \text{mode}(\{h_1(x), h_2(x), \ldots, h_T(x)\})$

For regression:

$\acute{y}\wedge = 1/T \sum_{t=1}^{T} h_t(x)$

🛠️ Steps:

Draw **bootstrap samples** from dataset.

Train a Decision Tree on each sample.

Use random subset of features at each split.

Aggregate predictions (majority vote or mean).

🔢 Example (Python):

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100)
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

Robust to overfitting.

Handles missing data well.

❌ Disadvantages:

Slower for large datasets.

Less interpretable than a single tree.

🎯 Use Cases:

Loan default prediction, stock trend analysis.

🔷 3. Support Vector Machine (SVM)

Definition: SVM finds a **hyperplane** that maximizes the margin between classes. For non-linear problems, it uses **kernel trick**.

🧮 Equation:

Hyperplane equation:

$f(x) = w^T x + b$

Optimization problem:

$\min 1/2||w||^2$ subject to $y_i(w^T x_i+b) \geq 1$

🛠 Steps:

Find hyperplane that maximizes margin.

Use **support vectors** to define boundary.

Apply kernels if data is not linearly separable.

🔢 Example (Python):

```
from sklearn.svm import SVC

model = SVC(kernel='rbf')
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

Effective in high dimensions.

Works with non-linear data using kernels.

❌ Disadvantages:

Slow with large datasets.

Needs parameter tuning (C, gamma).

🎯 Use Cases:

Face recognition, text classification.

🔷 4. K-Nearest Neighbors (KNN)

Definition: Instance-based algorithm. Predicts labels based on **k nearest neighbors**.

🧮 Equation (Euclidean Distance):

$d(x,y) = \sqrt{(\sum_{i=1}^{n}(x_i - y_i)^2)}$

🛠️ Steps:

Choose **k**.

Find nearest neighbors.

Classification: majority vote.

Regression: average value.

🔢 Example (Python):

```
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier(n_neighbors=3)
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

Simple, no training needed.

Works for multi-class problems.

❌ Disadvantages:

Slow for large datasets.

Sensitive to irrelevant features.

🎯 Use Cases:

Recommendation systems, image recognition.

🔷 5. Naïve Bayes

Definition: Probabilistic classifier based on **Bayes' Theorem**, assuming features are independent.

🧮 Equation:

P(Y|X) = (P(X|Y)P(Y))/P(X)

📌 Types:

Gaussian NB: Continuous features.

Multinomial NB: Text classification.

Bernoulli NB: Binary features.

🔢 Example (Python):

```
from sklearn.naive_bayes import GaussianNB

model = GaussianNB()
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

Fast and efficient.

Works well for text data.

❌ Disadvantages:

Strong independence assumption.

Poor if features are correlated.

🎯 Use Cases:

Spam detection, sentiment analysis.

🔷 6. Gradient Boosting

Definition: An ensemble technique that builds models sequentially, each correcting errors of the previous one.

🧮 Equation:

$F_m(x) = F_{m-1}(x) + \eta\, h_m(x)$

η: learning rate

$h_m(x)$: weak learner (small tree)

🛠️ Steps:

Train weak learner.

Compute residuals.

Fit next learner on residuals.

Combine predictions.

🔢 Example (Python):

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()
model.fit(X, y)
print(model.predict([X[0]]))
```

✅ Advantages:

High accuracy.

Works well for mixed data types.

❌ Disadvantages:

Requires careful tuning.

Slower to train.

🎯 Use Cases:

Credit scoring, Kaggle competitions.

🔥 Quick Comparison Table

Algorithm	Type	Strengths	Weaknesses
Decision Tree	Single model	Easy to interpret	Overfitting
Random Forest	Ensemble	Robust, high accuracy	Slower
SVM	Linear/Kernel	Works in high dimensions	Expensive on large data
KNN	Instance-based	Simple, non-parametric	Slow, sensitive to noise
Naïve Bayes	Probabilistic	Fast, good for text	Independence assumption
Gradient Boosting	Ensemble	High accuracy, flexible	Overfitting, slow