
📅 Day 8: Random Forests & Ensemble Methods

🎯 Objective

To understand how **ensemble learning**, particularly **Random Forests**, enhances model performance by combining multiple decision trees.

🌲 What is a Random Forest?

A **Random Forest** is:

- An **ensemble** of many **Decision Trees**
- Each tree makes a prediction; the **majority vote (classification)** or **average (regression)** is the final output
- It uses **random subsets of features and data** to build each tree

> 💡 It's like asking a group of friends for their opinions and taking a vote—better than trusting just one!

🤝 Why Use Ensemble Methods?

🔹 Weak Learners ➜ Strong Learner

- A single tree may **overfit** or perform poorly
- Many trees, when combined, **generalize better**

🔹 Reduces Variance

- Since each tree is trained on a different subset, the final model is more **stable and less sensitive** to noise

🔹 Handles High-Dimensional Data

- Random selection of features makes it robust to irrelevant variables

---

🔢 How Random Forests Work

1. **Bootstrap Sampling (Bagging)**:
    - Each tree is trained on a random **subset** of training data with replacement
2. **Random Feature Selection**:
    - At each node, a random subset of features is considered to split
3. **Prediction Aggregation**:
    - **Classification**: Most common class across trees
    - **Regression**: Average of all outputs

🧪 Step-by-Step Implementation

📦 Step 1: Load & Preprocess the Dataset

We use the **Breast Cancer Wisconsin dataset**:

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
```

```
X_test_scaled = scaler.transform(X_test)
```

🌳 Step 2: Train a Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report

rf_model = RandomForestClassifier(n_estimators=100, max_depth=6, random_state=42)
rf_model.fit(X_train_scaled, y_train)

y_pred = rf_model.predict(X_test_scaled)

print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred, target_names=data.target_names))
```

- `n_estimators=100`: 100 trees in the forest
- `max_depth=6`: Controls tree complexity to reduce overfitting

📊 Step 3: Feature Importance

This shows **which features were most helpful** in making decisions:

```
import matplotlib.pyplot as plt
import numpy as np

importances = rf_model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(12, 6))
plt.title('Feature Importances')
plt.bar(range(X.shape[1]), importances[indices])
plt.xticks(range(X.shape[1]), df.columns[indices], rotation=90)
plt.tight_layout()
plt.show()
```

🔁 Step 4: Experiment with Hyperparameters

You can try different combinations of trees, depth, and criteria:

```
rf_model_alt = RandomForestClassifier(n_estimators=50, max_depth=3, random_state=42)
rf_model_alt.fit(X_train_scaled, y_train)
alt_pred = rf_model_alt.predict(X_test_scaled)
```

```
print("Accuracy:", accuracy_score(y_test, alt_pred))
```

📋 Comparison Table

Model	Overfitting Risk	Interpretability	Accuracy	Stability
Decision Tree	High	Easy	Moderate	Low
Random Forest	Low	Medium	High	High

✅ Summary

- ✅ Random Forests are **ensemble models** that improve accuracy and reduce overfitting.
- ✅ They aggregate predictions from multiple trees.
- ✅ Important to tune `n_estimators`, `max_depth`, and `max_features`.
- ✅ Feature importance gives insights into key variables.

📅 Day 9: Cross-Validation & Hyperparameter Tuning

🎯 Goal

To make your machine learning models **more robust and accurate** by:

- Evaluating them more reliably using **Cross-Validation**
- Optimizing their configuration using **Hyperparameter Tuning**

🔄 What is Cross-Validation?

✅ Problem with Train-Test Split:

When we use only one train/test split, the performance might:

- Depend too much on **how the data was split**
- Give misleading results due to **randomness** or **bias**

🧪 Cross-Validation Solution:

K-Fold Cross-Validation splits data into **K parts (folds)**:

- Model trains on `K-1` parts and tests on the `1` remaining part
- This repeats **K times**, with each part used as a test set once
- The final score is the **average performance** across all K runs

Example (K=5):
Fold 1 → test, Fold 2-5 → train
Fold 2 → test, Fold 1,3-5 → train
... repeat 5 times

⚙️ What is Hyperparameter Tuning?

📌 What are Hyperparameters?

- Parameters **not learned** from data but **set manually**
- Examples for Random Forest:
    - `n_estimators` (number of trees)
    - `max_depth` (depth of tree)
    - `criterion` (gini/entropy)

🎯 Goal of Tuning:

To find the **best combination of hyperparameters** that gives the highest cross-validation score.

🧪 Tuning Techniques

1. 🔍 Grid Search (GridSearchCV)

- Tries **every combination** of values in your parameter grid
- Can be slow with many options

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'criterion': ['gini', 'entropy']
}

grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
```

2. 🎲 Randomized Search (RandomizedSearchCV)

- Picks a **random sample of combinations** from a given range
- Faster and effective for large grids

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

param_dist = {
    'n_estimators': randint(50, 200),
    'max_depth': randint(2, 10)
}

random_search = RandomizedSearchCV(RandomForestClassifier(), param_dist, n_iter=10,
cv=5)
```

💡 How to Apply This in Practice

🔹 Step-by-step:

1. **Split and scale data**

2. **Define model + hyperparameter space**
3. **Use GridSearchCV or RandomizedSearchCV**
4. **Train on best model**
5. **Evaluate on test data**

🧪 Example Using Breast Cancer Dataset

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load and prepare data
data = load_breast_cancer()
X, y = data.data, data.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Grid Search
param_grid = {
    'n_estimators': [50, 100, 150],
    'max_depth': [3, 5, 7],
    'criterion': ['gini', 'entropy']
}

grid = GridSearchCV(RandomForestClassifier(), param_grid, cv=5)
grid.fit(X_train_scaled, y_train)

print("Best Params:", grid.best_params_)

# Evaluate
from sklearn.metrics import classification_report
y_pred = grid.best_estimator_.predict(X_test_scaled)
print(classification_report(y_test, y_pred))
```

✅ Summary Table

Concept	Description
Cross-Validation	Tests model on different data splits
K-Fold CV	Common CV strategy with k partitions
GridSearchCV	Tests all param combinations (exhaustive)
RandomizedSearchCV	Tests random param combinations (faster)
Best Practice	Always tune models with CV, not just 1 split

👩‍💻 Intern Challenge

- 🔧 Tune a DecisionTreeClassifier
- Compare performance with and without CV
- Try RandomizedSearchCV on Logistic Regression

📅 Day 10: Logistic Regression & ROC Curve

🎯 Objective

Learn how to:

- Build and evaluate a **Logistic Regression model**
- Understand its probabilistic nature
- Use **ROC Curve** and **AUC Score** for performance evaluation

---

🔍 What is Logistic Regression?

Despite its name, **Logistic Regression is a classification algorithm**, not regression.

📘 Key Points:

- It models the **probability** that a given input belongs to a class.
- Uses the **sigmoid function** to squeeze output between **0 and 1**.
- Ideal for **binary classification problems** like:
  "Spam or Not", "Tumor is Benign or Malignant", "Customer will Buy or Not".

⚙️ Equation:

P(y=1∣X)=11+e−z,where z=wX+bP(y=1|X) = \frac{1}{1 + e^{-z}}, \quad \text{where } z = wX + b

---

🧪 Step-by-Step Implementation

📦 Step 1: Load and Preprocess Data

We'll use the **Breast Cancer Wisconsin dataset**.

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd

# Load dataset
data = load_breast_cancer()
df = pd.DataFrame(data.data, columns=data.feature_names)
df['target'] = data.target

# Split data
X = df.drop('target', axis=1)
y = df['target']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)

# Scale data
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

🤖 Step 2: Train the Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
```

```
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

y_pred = model.predict(X_test_scaled)
print("Accuracy:", accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

🔍 Step 3: Evaluate with ROC Curve & AUC

📘 **ROC Curve (Receiver Operating Characteristic):**

- Shows tradeoff between **TPR (Recall)** and **FPR (False Positives)**
- More area under the curve = Better classifier

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities
y_prob = model.predict_proba(X_test_scaled)[:, 1]

# Calculate ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

# Plot
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f"ROC Curve (AUC = {auc:.2f})")
plt.plot([0, 1], [0, 1], 'k--')  # baseline
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.grid(True)
plt.show()
```

📊 Interpretation

Metric	Meaning
Accuracy	% of correct predictions overall
Precision	How many predicted positives were correct

Recall	How many actual positives were detected
AUC	Overall quality of model's probability ranking

✅ Summary

- Logistic Regression is a **probabilistic classifier** using the sigmoid function
- Predicts classes based on a **threshold** (default = 0.5)
- ROC Curve and AUC are crucial for evaluating **probabilistic models**

👩‍💻 Intern Challenge

- Try using logistic regression on the **Titanic dataset**
- Adjust the threshold (e.g., 0.3, 0.7) and observe how precision/recall change
- Compare Logistic Regression vs Decision Tree on the same dataset

---

📅 Day 11: KNN & SVM (K-Nearest Neighbors and Support Vector Machines)

🎯 Objective

By the end of this day, interns will:

- Understand the concepts behind **KNN** and **SVM**
- Learn to train, test, and evaluate both models

- Compare their performance using **ROC Curve** and **AUC Score**

🧠 1. What is K-Nearest Neighbors (KNN)?

KNN is a **lazy learning algorithm**:

- It doesn't learn a model from the training data
- Instead, it **stores the entire dataset** and makes predictions only when asked

🔍 How it works:

1. Choose a number K (e.g., 3, 5)
2. To predict a new point, find the K nearest points in the training set using **Euclidean distance**
3. Return the **majority class** among those neighbors

✅ Pros:

- Simple, intuitive, no training
- Good for small datasets

⚠️ Cons:

- Slow with large datasets
- Sensitive to feature scaling and irrelevant features

💡 2. What is Support Vector Machine (SVM)?

SVM is a **supervised machine learning algorithm** that works by:

- Finding the **best hyperplane** that separates data into different classes
- The "best" means the one with the **maximum margin**

🧾 Key Concepts:

- **Support Vectors**: Data points closest to the decision boundary
- **Kernel Trick**: SVM can project data into higher dimensions using kernels (linear, polynomial, RBF)

✅ Pros:

- Works well in high-dimensional spaces
- Robust to outliers
- Effective when number of features > number of samples

⚠️ Cons:

- Can be slower with very large datasets
- Requires careful tuning of kernel and hyperparameters

🧪 Step-by-Step Implementation

We use the **Breast Cancer dataset** (binary classification).

🔹 Step 1: Load and Preprocess Data

```
from sklearn.datasets import load_breast_cancer

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler


data = load_breast_cancer()

X, y = data.data, data.target


# Train-test split
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,
random_state=42)


# Standardize

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

🤖 Step 2: KNN Model

```
from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy_score, classification_report


knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train_scaled, y_train)


y_pred_knn = knn.predict(X_test_scaled)

print("KNN Accuracy:", accuracy_score(y_test, y_pred_knn))

print(classification_report(y_test, y_pred_knn))
```

- `n_neighbors=5` means using the 5 nearest data points
- Accuracy & classification report gives precision, recall, f1-score

---

🧠 Step 3: SVM Model

```
from sklearn.svm import SVC
```

```
svm = SVC(kernel='linear', probability=True)
svm.fit(X_train_scaled, y_train)

y_pred_svm = svm.predict(X_test_scaled)
print("SVM Accuracy:", accuracy_score(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

- `kernel='linear'` means we're using a simple linear decision boundary
- `probability=True` is required to generate ROC curve

📊 Step 4: Compare with ROC Curve and AUC

```
from sklearn.metrics import roc_curve, roc_auc_score
import matplotlib.pyplot as plt

# Predict probabilities
y_proba_knn = knn.predict_proba(X_test_scaled)[:, 1]
y_proba_svm = svm.predict_proba(X_test_scaled)[:, 1]

# ROC data
fpr_knn, tpr_knn, _ = roc_curve(y_test, y_proba_knn)
fpr_svm, tpr_svm, _ = roc_curve(y_test, y_proba_svm)
auc_knn = roc_auc_score(y_test, y_proba_knn)
auc_svm = roc_auc_score(y_test, y_proba_svm)

# Plot ROC
```

```
plt.figure(figsize=(8, 6))
plt.plot(fpr_knn, tpr_knn, label=f'KNN (AUC = {auc_knn:.2f})')
plt.plot(fpr_svm, tpr_svm, label=f'SVM (AUC = {auc_svm:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve Comparison")
plt.legend()
plt.grid(True)
plt.show()
```

📋 Comparison Table

Model	Learning Type	Training Time	Accuracy	Strengths
KNN	Lazy	Fast	Good	Simple, interpretable
SVM	Eager	Slower	Better	Robust, handles high-dimensions

✅ Summary

- **KNN** classifies based on nearby points
- **SVM** finds the best boundary for classification
- Always evaluate using **AUC & ROC**, not just accuracy

- Scaling is important for both

👩‍💻 Intern Tasks

1. Try different values of `n_neighbors` in KNN and plot accuracy.
2. Switch SVM to `kernel='rbf'` and compare results.
3. Plot decision boundaries using 2 features (e.g., PCA).
