□ Day 8 – Decision Tree & Random Forest Classification

Session Summary

On Day 8 of our AI/ML journey, we focused on two very important tree-based classification techniques:

- **Decision Tree Classifier**
- **Random Forest Classifier**

Before diving into model building, we performed data analysis and visualization on the **Iris dataset** using libraries like **Seaborn, Pandas, and Matplotlib**. We also cleaned the data to ensure high model performance.

Decision Tree – A Simple Yet Powerful Classifier

A **Decision Tree** works like a real decision-making flowchart. It's a supervised learning algorithm that can be used for both **classification and regression** tasks.

How It Works:

- Internal **nodes** represent features (like petal length)
- **Branches** represent conditions or decisions
- **Leaf nodes** hold the final class label (species in our case)

The algorithm splits the dataset using a criterion like **Gini Index** or **Information Gain**, trying to reduce impurity at each level.

□ Key Characteristics:

- Easy to understand and visualize
- Can handle both numerical and categorical data
- But... it's prone to **overfitting**, especially on small or noisy datasets

---

Random Forest – A Smarter Ensemble Approach

While a Decision Tree is interpretable, it often fails to generalize well. That's where **Random Forest** comes in.

It builds **multiple decision trees**, each trained on a random subset of the data (this is called **Bagging**). The final output is determined by taking a **majority vote** from all trees.

Why It's Better:

- More robust and accurate
- Handles overfitting effectively

- Works well even with missing data or unbalanced classes

Data Preparation & Cleaning

We worked with the classic **Iris dataset**, which includes:

- 4 features: sepal_length, sepal_width, petal_length, petal_width
- 3 classes of flowers: Setosa, Versicolor, Virginica

🛠 **Steps Performed:**

df = pd.read_csv("Iris.csv")

df.drop_duplicates(inplace=True)

df.isna().sum() # Checked for missing values

After cleaning, we explored the dataset with:

- .info() and .describe() for understanding structure and stats
- .head() to preview top records

📊 Data Visualization

We used **Seaborn** to understand the data better before training our models.

👁 **Visuals Created:**

- **Countplot** to see the number of samples per species
- **Barplot** comparing mean sepal lengths
- **Boxplot** to detect spread and potential outliers

These visualizations helped us verify class balance and feature variance.

Model Building & Evaluation

Step 1: Feature & Label Preparation

X = df[['sepal_length', 'sepal_width', 'petal_length', 'petal_width']]

y = df['species']

y_encoded = LabelEncoder().fit_transform(y)

Step 2: Splitting the Data

x_train, x_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

🌳 Decision Tree Classifier

dt = DecisionTreeClassifier(random_state=42)

dt.fit(x_train, y_train)

y_pred = dt.predict(x_test)

✅ Accuracy:

accuracy_score(y_test, y_pred) # Output: 1.0

📊 Confusion Matrix:

We used a heatmap to visualize prediction performance:

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, cmap='Blues')

🌲 Random Forest Classifier

rf = RandomForestClassifier(random_state=42)

rf.fit(x_train, y_train)

y_pred_rf = rf.predict(x_test)

✅ Accuracy:

accuracy_score(y_test, y_pred_rf) # Output: 1.0

📊 Confusion Matrix:

sns.heatmap(confusion_matrix(y_test, y_pred_rf), annot=True, cmap='Greens')

□ Final Observations

- Both models achieved **100% accuracy** on the Iris dataset, mainly because it's well-structured, balanced, and not very complex.
- **Decision Tree** worked well but may overfit on larger or noisy datasets.
- **Random Forest** gave the same accuracy but is more reliable due to ensemble voting.
- Visualization via confusion matrix made it easier to understand model performance.

📝 Conclusion

✅ Learned how **Decision Trees** make step-by-step decisions based on feature splits
✅ Understood how **Random Forest** aggregates multiple trees to enhance accuracy and reduce variance
✅ Applied both models effectively on a real-world dataset
✅ Practiced full ML pipeline: from data cleaning and visualization to modeling and evaluation