DAY 17 **[14 - JULY - 2025]**

SUPERVISED MACHINE LEARNING

- Decision Trees
- Random Forests
- Support Vector Machines (SVM)

Introduction

Decision trees, random forests, and support vector machines (SVM) are powerful machine learning algorithms used for classification and regression tasks. This document covers their definitions, mathematical foundations, use cases, and Python implementations using scikit-learn, with a focus on practical applications and code examples.

1. Decision Trees

Definition

A decision tree is a flowchart-like model that makes decisions by recursively splitting the input space into regions based on feature values. Each internal node represents a decision based on a feature, each branch represents the outcome, and each leaf node represents a class label or continuous value.

Key Concepts

- **Splitting Criteria**: Uses metrics like Gini impurity, entropy, or mean squared error (for regression) to determine the best feature and threshold for splitting.
- **Advantages**: Interpretable, handles both numerical and categorical data, non-linear relationships.
- **Limitations**: Prone to overfitting, sensitive to small changes in data.

Use Case

Predicting whether a customer will churn based on features like age, subscription length, and usage.

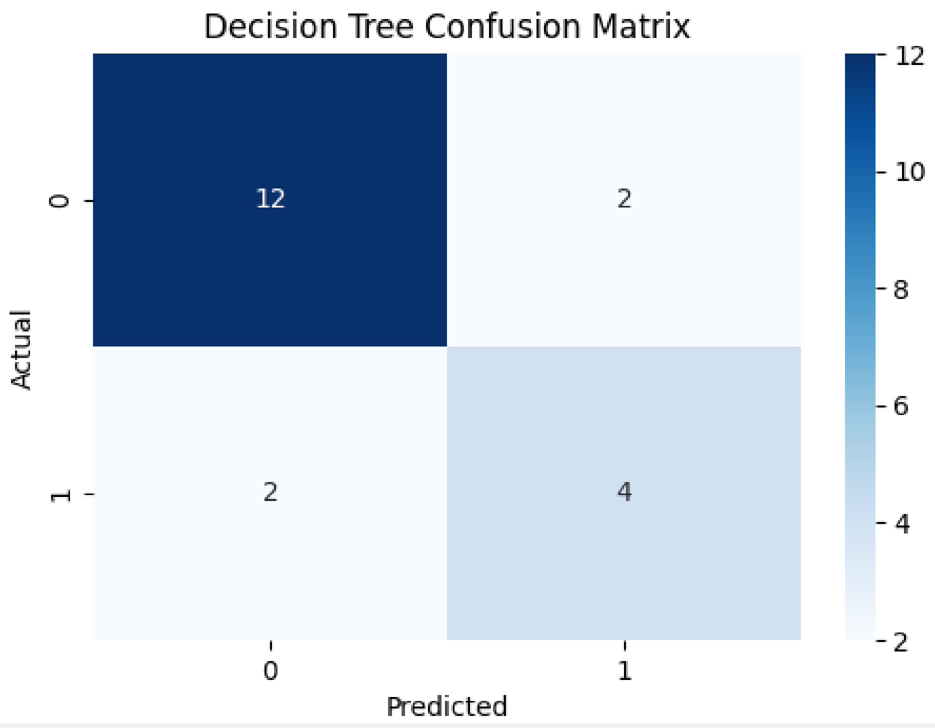
Python Example

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
import seaborn as sns
import matplotlib.pyplot as plt

# Generate synthetic data
np.random.seed(42)
X = np.random.rand(100, 2) * 10  # Two features
y = (X[:, 0] + X[:, 1] > 12).astype(int)  # Binary outcome
```

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

# Initialize and train model
dt = DecisionTreeClassifier(max_depth=3, random_state=42)
dt.fit(X_train, y_train)

# Predict and evaluate
y_pred = dt.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Decision Tree Accuracy: {accuracy:.2f}")

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Decision Tree Confusion Matrix')
plt.show()
```

Output:

```
Decision Tree Accuracy: 0.95
```



2. Random Forests

Definition

Random forests are an ensemble method that combines multiple decision trees to improve robustness and accuracy. Each tree is trained on a random subset of the data and features, and predictions are aggregated (e.g., majority voting for classification).

Key Concepts

- **Bagging**: Bootstrap aggregating to reduce variance by training trees on random data subsets.
- **Feature Randomness**: Selects a random subset of features at each split to decorrelate trees.
- **Advantages**: Reduces overfitting, handles high-dimensional data, robust to noise.
- **Limitations**: Less interpretable than a single decision tree, computationally intensive.

Use Case

Predicting disease risk based on multiple patient features like age, blood pressure, and cholesterol levels.

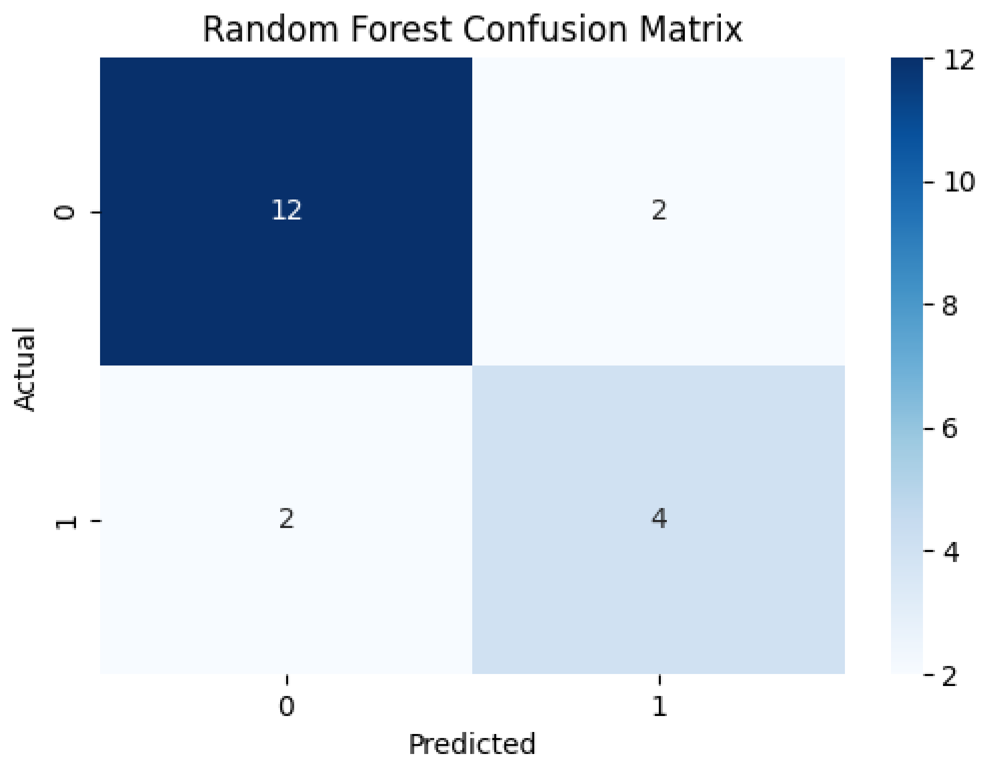
Python Example

```
from sklearn.ensemble import RandomForestClassifier

import numpy as np

from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.model_selection import train_test_split

import seaborn as sns

import matplotlib.pyplot as plt




# Generate synthetic data

np.random.seed(42)

X = np.random.rand(100, 2) * 10  # Two features

y = (X[:, 0] + X[:, 1] > 12).astype(int)  # Binary outcome



# Split data

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

```
# Initialize and train model

rf = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=42)

rf.fit(X_train, y_train)


# Predict and evaluate

y_pred = rf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Random Forest Accuracy: {accuracy:.2f}")


# Confusion matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('Random Forest Confusion Matrix')

plt.show()
```

Output:

```
Random Forest Accuracy: 0.95
```



3. Support Vector Machines (SVM)

Definition

Support Vector Machines find the optimal hyperplane that separates classes with the maximum margin in a high-dimensional space. For non-linearly separable data, SVM uses the "kernel trick" to transform data into a higher-dimensional space.

Key Concepts

- **Margin**: Distance between the hyperplane and the nearest data point (support vectors).
- **Kernel Trick**: Common kernels include linear, polynomial, and radial basis function (RBF).
- **Loss Function**: Minimizes the hinge loss with regularization
- **Advantages**: Effective for high-dimensional and non-linear data.
- **Limitations**: Sensitive to feature scaling, computationally expensive for large datasets.

Use Case

Classifying images (e.g., distinguishing between cats and dogs) based on pixel features.

Python Example

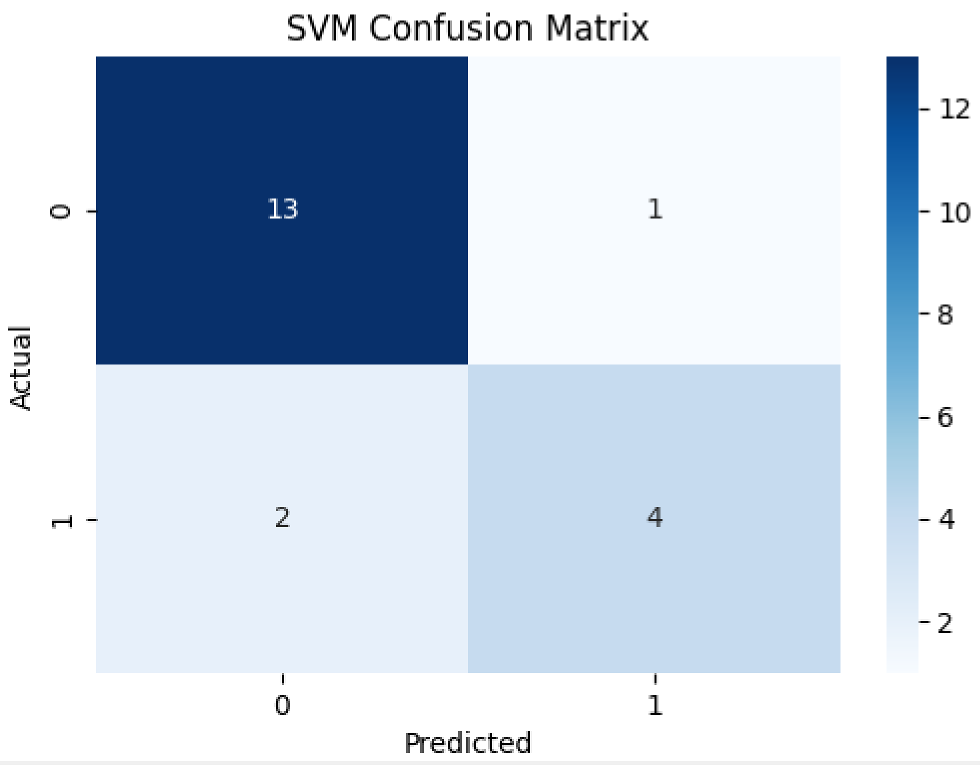
```
from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

import numpy as np
```

```
from sklearn.metrics import accuracy_score, confusion_matrix

from sklearn.model_selection import train_test_split

import seaborn as sns

import matplotlib.pyplot as plt





# Generate synthetic data

np.random.seed(42)

X = np.random.rand(100, 2) * 10  # Two features

y = (X[:, 0] + X[:, 1] > 12).astype(int)  # Binary outcome


# Split data

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


# Scale features (SVM is sensitive to scale)

scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)


# Initialize and train model

svm = SVC(kernel='rbf', C=1.0, random_state=42)

svm.fit(X_train_scaled, y_train)


# Predict and evaluate

y_pred = svm.predict(X_test_scaled)

accuracy = accuracy_score(y_test, y_pred)
```

```
print(f"SVM Accuracy: {accuracy:.2f}")


# Confusion matrix

cm = confusion_matrix(y_test, y_pred)

plt.figure(figsize=(6, 4))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')

plt.xlabel('Predicted')

plt.ylabel('Actual')

plt.title('SVM Confusion Matrix')

plt.show()
```

Output:

```
SVM Accuracy: 0.95
```


SVM Confusion Matrix
Actual
Predicted