1. **Write a program to classify a given dataset using the support vector machine.**

# Import necessary libraries

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.svm import SVC

from sklearn.metrics import accuracy\_score, classification\_report

from sklearn.preprocessing import StandardScaler

# Load dataset (replace with your dataset)

# For demonstration, we'll use the Iris dataset from scikit-learn

from sklearn.datasets import load\_iris

data = load\_iris()

# Extract features and labels

X = data.data

y = data.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42, stratify=y)

# Standardize the features (important for SVM performance)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create the SVM model (you can experiment with different kernels: 'linear', 'rbf', 'poly', etc.)

svm\_model = SVC(kernel='rbf', C=1.0, gamma='scale', random\_state=42)

# Train the SVM model

svm\_model.fit(X\_train, y\_train)

# Make predictions on the test set

y\_pred = svm\_model.predict(X\_test)

# Evaluate the model

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

print("\nClassification Report:\n", classification\_report(y\_test, y\_pred, target\_names=data.target\_names))

# Optional: Make a prediction for new data

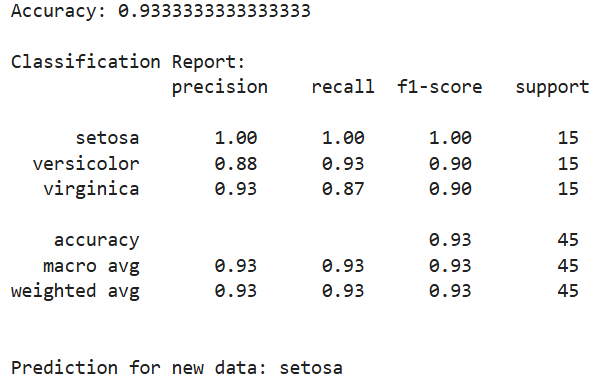
new\_data = np.array([[5.1, 3.5, 1.4, 0.2]]) # Replace with your own data

new\_data\_scaled = scaler.transform(new\_data)

prediction = svm\_model.predict(new\_data\_scaled)

print("\nPrediction for new data:", data.target\_names[prediction[0]])

**OUTPUT:**



1. **Write a program to demonstrate ensemble learning with any classifier on a given dataset.**

#pip install numpy pandas scikit-learn matplotlib

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification\_report, confusion\_matrix, accuracy\_score

# Load the Iris dataset

iris = load\_iris()

X = iris.data

y = iris.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a Random Forest Classifier

rf\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the classifier

rf\_classifier.fit(X\_train, y\_train)

# Make predictions

y\_pred = rf\_classifier.predict(X\_test)

# Evaluate the model

print("Confusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("Accuracy Score:", accuracy\_score(y\_test, y\_pred))

# Feature Importance

feature\_importances = rf\_classifier.feature\_importances\_

features = iris.feature\_names

# Plotting feature importances

plt.figure(figsize=(10, 6))

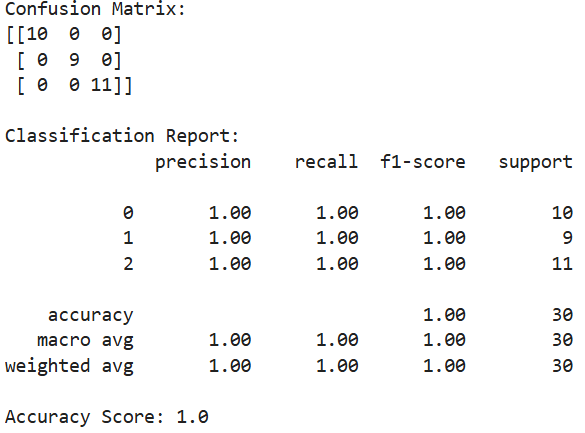
plt.barh(features, feature\_importances, color='skyblue')

plt.xlabel('Feature Importance')

plt.title('Feature Importance in Random Forest Classifier')

plt.show()

**OUTPUT:**



A blue rectangular object with white text

Description automatically generated

1. **Apply EM algorithm to cluster a set of data stored in a .CSV file. Use the same data set for clustering. using the k-Means algorithm. Compare the results of these two algorithms and comment on the quality of clustering. You can add Java/Python ML library classes/API in the program.**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.cluster import KMeans

from sklearn.metrics import accuracy\_score, silhouette\_score

iris = load\_iris()

x = pd.DataFrame(iris.data, columns=iris.feature\_names)

y = iris.target

plt.figure(figsize=(20,7))

plt.subplot(1, 2, 1)

plt.scatter(x['sepal length (cm)'], x['sepal width (cm)'], c=y, s=40)

plt.title('Sepal')

plt.subplot(1, 2, 2)

plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=y, s=40)

plt.title('Petal')

model = KMeans(n\_clusters=3)

model.fit(x)

model.labels\_

accuracy\_score(y, model.labels\_)

from sklearn.mixture import GaussianMixture

GMM = GaussianMixture(n\_components=3) # Instantiate and fit the model

GMM.fit(x)

gmm\_clusters = GMM.predict(x)

gmm\_clusters

accuracy\_score(y, gmm\_clusters)

plt.figure(figsize=(17,7))

plt.subplot(1, 3, 1)

plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=y, s=40)

plt.title('Actual Classes')

plt.subplot(1, 3, 2)

plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=model.labels\_,

s=40)

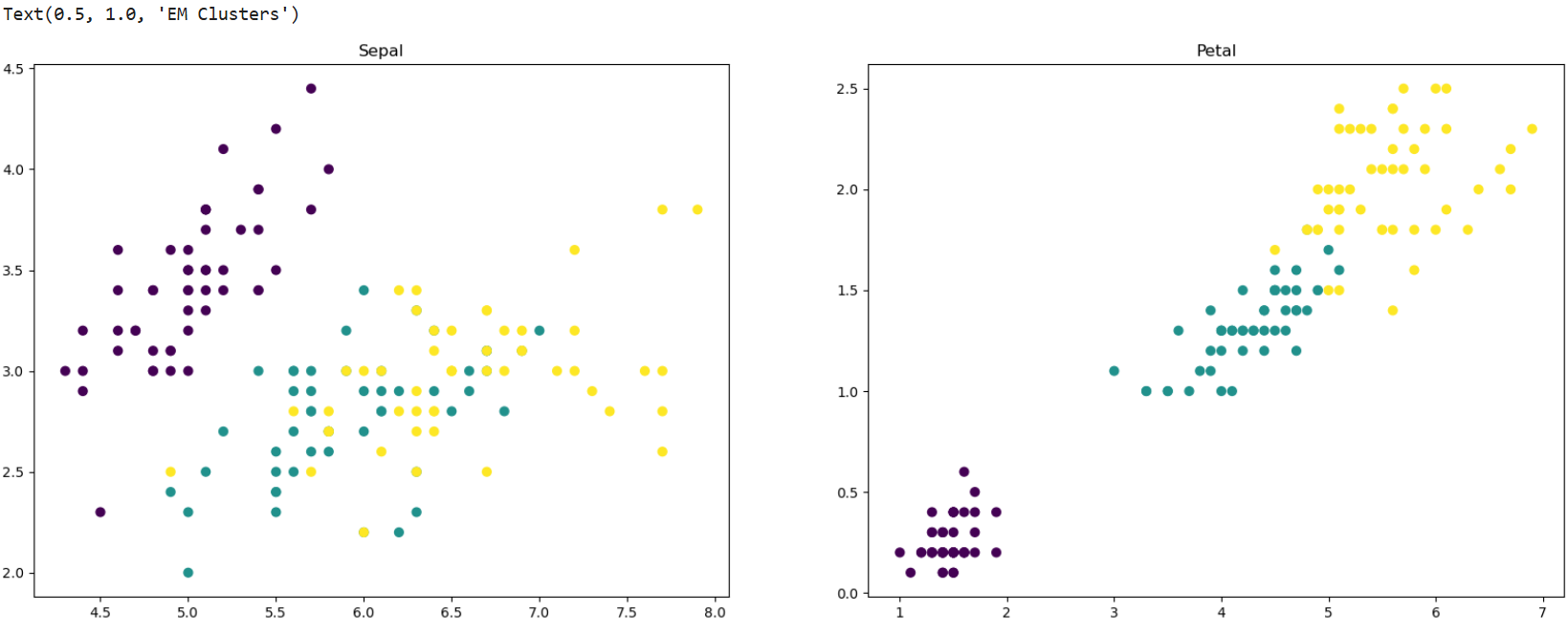
plt.title('KMeans Clusters')

plt.subplot(1, 3, 3)

plt.scatter(x['petal length (cm)'], x['petal width (cm)'], c=gmm\_clusters, s=40)

plt.title('EM Clusters')

**OUTPUT :**



A graph of a graph

Description automatically generated with medium confidence