**Lab 6:**

Program 1: Write a program to perform CART and ID3 decision tree classifier on iris data.

# Import necessary libraries  
**from** sklearn.datasets import load\_iris  
**from** sklearn.model\_selection import train\_test\_split  
**from** sklearn.tree import DecisionTreeClassifier  
**from** sklearn.metrics import accuracy\_score  
**from** sklearn import tree  
# Load the Iris dataset (can be replaced with any dataset)  
data = load\_iris()  
X = data.data  
y = data.target  
  
# Split the dataset into training and testing sets (80% train, 20% test)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, **test\_size**=0.2, **random\_state**=42)  
  
***# Initialize the Decision Tree Classifier (CART)***  
clf = DecisionTreeClassifier(**criterion**='gini', **random\_state**=42)  
  
# Train the model using the training data  
clf.fit(X\_train, y\_train)  
  
# Make predictions using the test data  
y\_pred = clf.predict(X\_test)  
  
# Calculate and print the accuracy  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {accuracy \* 100:.2f}%")  
  
# Visualize the Decision Tree  
import matplotlib.pyplot as plt  
  
plt.figure(figsize=(12, 8))  
tree.plot\_tree(clf, **filled**=True, **feature\_names**=data.feature\_names, **class\_names**=data.target\_names, **rounded**=True)  
plt.show()  
**from** sklearn.tree import export\_text  
***# Train the Decision Tree Classifier (ID3)***  
model = DecisionTreeClassifier(**criterion**="entropy", **random\_state**=42)  
model.fit(X\_train, y\_train)  
  
# Predict on test data  
y\_pred = model.predict(X\_test)  
  
# Evaluate the model  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {accuracy:.2f}")  
plt.figure(figsize=(12, 8))  
tree.plot\_tree(model, **filled**=True, **feature\_names**=data.feature\_names, **class\_names**=data.target\_names, **rounded**=True)  
plt.show()

**Lab 7:**

Program 1: Write a program to classify data using Naïve Bayes classification

%matplotlib inline  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns; sns.set()  
**from** sklearn.datasets import make\_blobs  
# make\_blobs is a dataset   
X, y = make\_blobs(100, 2, **centers**=2, **random\_state**=2, **cluster\_std**=1.5)  
plt.scatter(X[:, 0], X[:, 1], **c**=y, **s**=50, **cmap**='RdBu');  
plt.show()  
**from** sklearn.naive\_bayes import GaussianNB  
model = GaussianNB()  
model.fit(X, y);  
rng = np.random.RandomState(0)  
rng  
plt.scatter(X[:, 0], X[:, 1], **c**=y, **s**=50, **cmap**='RdBu')  
lim = plt.axis()  
plt.scatter(Xnew[:, 0], Xnew[:, 1], **c**=ynew, **s**=20, **cmap**='RdBu', **alpha**=0.1)  
plt.axis(lim);  
plt.show()  
yprob = model.predict\_proba(Xnew)  
yprob[-8:].round(2)

**Lab 8:**

Program 1: Write a program to classify data using Support Vector Machine

# Import necessary libraries  
**from** sklearn **import** datasets  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn.svm **import** SVC  
**from** sklearn.metrics **import** accuracy\_score, classification\_report, confusion\_matrix  
iris = datasets.load\_iris()  
X = iris.data  
y = iris.target  
print("X",X)  
print("y",y)  
# Split the dataset into training and testing sets (80% train, 20% test)  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
# Initialize the Support Vector Classifier (SVM)  
svm\_model = SVC(kernel='linear') # You can use 'rbf', 'poly', or 'sigmoid' for different kernels  
  
# Train the SVM model using the training data  
svm\_model.fit(X\_train, y\_train)  
  
# Make predictions on the test data  
y\_pred = svm\_model.predict(X\_test)  
  
# Calculate and print the accuracy  
accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {accuracy \* 100:.2f}%")  
  
# Print the classification report and confusion matrix for detailed evaluation  
print("\nClassification Report:")  
print(classification\_report(y\_test, y\_pred, target\_names=iris.target\_names))  
  
print("Confusion Matrix:")  
print(confusion\_matrix(y\_test, y\_pred))

**Lab 9:**

Program 1: Write a program to classify data using K-Nearest Neighbor classification

import numpy as np   
import pandas as pd   
**from** sklearn.model\_selection import train\_test\_split   
**from** sklearn.neighbors import KNeighborsClassifier   
import matplotlib.pyplot as plt   
import seaborn as sns  
**from** sklearn.datasets import load\_iris  
**from** sklearn.model\_selection import train\_test\_split  
  
iris = load\_iris(**as\_frame**=True)  
X = iris.data[["sepal length (cm)", "sepal width (cm)"]]  
y = iris.target  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, **stratify**=y, **random\_state**=0)  
**from** sklearn.neighbors import KNeighborsClassifier  
**from** sklearn.pipeline import Pipeline  
**from** sklearn.preprocessing import StandardScaler  
  
clf = Pipeline( steps=[("scaler", StandardScaler()), ("knn", KNeighborsClassifier(**n\_neighbors**=11))])  
import matplotlib.pyplot as plt  
  
**from** sklearn.inspection import DecisionBoundaryDisplay  
  
\_, axs = plt.subplots(**ncols**=2, figsize=(12, 5))  
  
**for** ax, weights **in** zip(axs, ("uniform", "distance")):  
 clf.set\_params(**knn\_\_weights**=weights).fit(X\_train, y\_train)  
 disp = DecisionBoundaryDisplay.from\_estimator(  
 clf,  
 X\_test,  
 **response\_method**="predict",  
 **plot\_method**="pcolormesh",  
 **xlabel**=iris.feature\_names[0],  
 **ylabel**=iris.feature\_names[1],  
 **shading**="auto",  
 **alpha**=0.5,  
 **ax**=ax,  
 )  
 scatter = disp.ax\_.scatter(X.iloc[:, 0], X.iloc[:, 1], **c**=y, **edgecolors**="k")  
 disp.ax\_.legend(  
 scatter.legend\_elements()[0],  
 iris.target\_names,  
 **loc**="lower left",  
 **title**="Classes",  
 )  
 \_ = disp.ax\_.set\_title(  
 f"3-Class classification\n(k={clf[-1].n\_neighbors}, weights={weights!r})")  
plt.show()

**Lab 10:**

Program 1: Write a program to perform implement K-means clustering on given data

#implementation K-means clustering python  
#importing useful libraries  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
from sklearn.cluster import KMeans  
from sklearn import datasets  
#loading iris data using sklearn library  
iris = datasets.load\_iris()  
  
#splitting data **in** two different variables, containing data and target respectively.  
x = pd.DataFrame(iris.data)  
x.columns = ['Sepal\_Length','Sepal\_width','Petal\_Length','Petal\_width']  
  
y = pd.DataFrame(iris.target)  
y.columns = ['Targets']  
#preparing the model  
model=KMeans(n\_clusters=2)  
  
#training the model  
model.fit(x)  
#plotting the scatter points of the unlabelled data  
plt.scatter(x.Petal\_Length, x.Petal\_width)  
plt.show()  
  
#creating **a** **color** map to assign different colors to different clusters  
colormap=np.array(['Red','green','blue'])  
  
#plotting the classified points  
plt.scatter(x.Petal\_Length, x.Petal\_width,c=colormap[y.Targets],s=40)  
plt.title('Classification réelle')  
plt.show()  
  
#plot of the final clusters formed by the K-means algorithm  
plt.scatter(x.Petal\_Length, x.Petal\_width,c=colormap[model.labels\_],s=40)  
plt.title('Classification K-means ')  
plt.show()

**Lab 11: (This is additional Program)**

Program 1: Write a program to perform Hierarchical clustering on given data

**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**from** sklearn.datasets **import** make\_blobs  
**from** sklearn.**cluster** **import** AgglomerativeClustering  
**from** scipy.**cluster**.hierarchy **import** dendrogram, linkage  
# Generate sample data  
X, y = make\_blobs(n\_samples=300, centers=4, cluster\_std=0.60, random\_state=0)  
# **Perform** agglomerative clustering  
agg\_clustering = AgglomerativeClustering(n\_clusters=4)  
y\_pred = agg\_clustering.fit\_predict(X)  
# Plot the clusters  
plt.scatter(X[:, 0], X[:, 1], c=y\_pred, cmap='rainbow')  
plt.title('Agglomerative Clustering')  
plt.xlabel('Feature 1')  
plt.ylabel('Feature 2')  
plt.**show**()  
# Generate the linkage matrix  
Z = linkage(X, **method**='ward')  
  
# Plot the dendrogram  
plt.figure(figsize=(10, 7))  
dendrogram(Z)  
plt.title('Dendrogram')  
plt.xlabel('Sample Index')  
plt.ylabel('Distance')  
plt.**show**()