**Practical No: 08**

**Aim: Implementing ensemble Learning Algorithm and Support Vector Machine(SVM) Algorithm**

**1. Ensemble Learning**

import numpy as np

from sklearn import datasets

from sklearn import model\_selection

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

warnings.simplefilter("ignore")

iris = datasets.load\_iris()

x,y = iris.data,iris.target

labels = ["Random Forest","Logistic Regression","GaussianNB","Decision Tree"]

m1 = RandomForestClassifier(random\_state=42)

m2 = LogisticRegression(random\_state=1)

m3 = GaussianNB()

m4 = DecisionTreeClassifier()

for m,label in zip([m1,m2,m3,m4],labels):

scores = model\_selection.cross\_val\_score(m,x,y,cv=5,scoring="accuracy")

print(f"Accuracy: {scores.mean()} {label}")

voting\_clf\_hard = VotingClassifier(estimators=[(labels[0],m1),(labels[1],m2),(labels[2],m3),(labels[3],m4)],voting='hard')

voting\_clf\_soft = VotingClassifier(estimators=[(labels[0],m1),(labels[1],m2),(labels[2],m3),(labels[3],m4)],voting='soft')

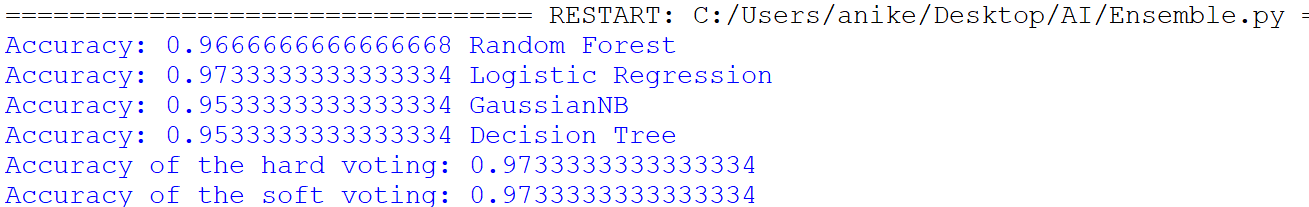
scores1 = model\_selection.cross\_val\_score(voting\_clf\_hard,x,y,cv=5,scoring="accuracy")

scores2 = model\_selection.cross\_val\_score(voting\_clf\_soft,x,y,cv=5,scoring="accuracy")

print(f"Accuracy of the hard voting: {scores1.mean()}")

print(f"Accuracy of the soft voting: {scores2.mean()}")

**Output:**



**2. Support Vector Machine**

import numpy as np

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.svm import SVC

from sklearn.preprocessing import StandardScaler

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

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data[:, :2] # Using only the first two features (sepal length and sepal width)

y = iris.target

# Split the data 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)

# Standardize the features

scaler = StandardScaler()

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

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

# Train the SVM model with a linear kernel

svm\_model = SVC(kernel='linear')

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

# Function to plot decision boundaries

def plot\_decision\_boundaries(X, y, model):

x\_min, x\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

y\_min, y\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx, yy = np.meshgrid(np.arange(x\_min, x\_max, 0.01), np.arange(y\_min, y\_max, 0.01))

Z = model.predict(np.c\_[xx.ravel(), yy.ravel()])

Z = Z.reshape(xx.shape)

plt.contourf(xx, yy, Z, alpha=0.3)

plt.scatter(X[:, 0], X[:, 1], c=y, edgecolors='k', marker='o', s=100, cmap=plt.cm.Paired)

plt.xlabel('Sepal length (standardized)')

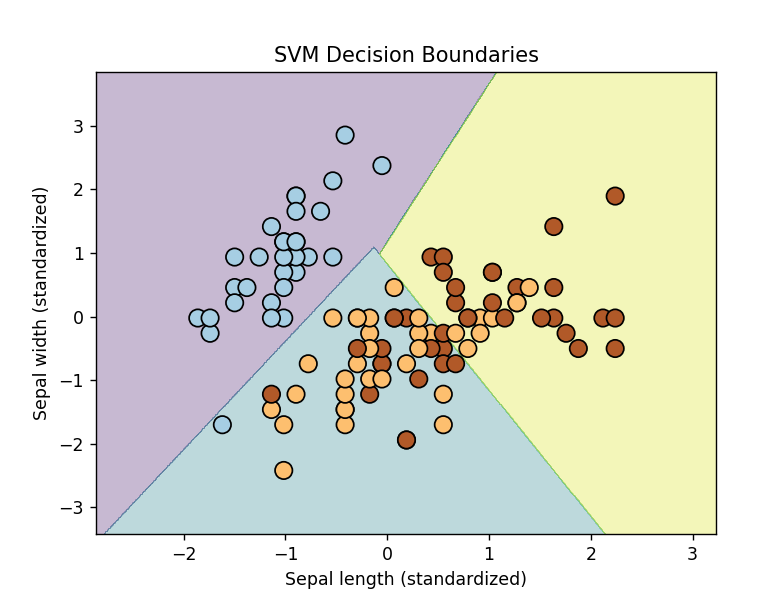
plt.ylabel('Sepal width (standardized)')

plt.title('SVM Decision Boundaries')

plt.show()

# Plot decision boundaries using the training set

plot\_decision\_boundaries(X\_train, y\_train, svm\_model)

**Output:**