SSN COLLEGE OF ENGINEERING (Autonomous)
Affiliated to Anna University
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Subject: Machine Learning Lab

Class: CSE B **Acad. Year:** 2023-2024

Date: 02-05-2024

Name: Roshni Badrinath **Reg no.:** 3122 21 5001 087

ML LAB TEST 1

AIM:
To develop a python program to predict the type of raisin using all the classification models (LR, PLA, MLP, KNN, SVM, Naïve Bayes) learnt and interpret the model that works better for the given dataset as well as visualize the features from the dataset and interpret the results obtained by the model using the Matplotlib library.

Dataset:
https://archive.ics.uci.edu/dataset/850/raisin

Dataset contains images of Kecimen and Besni raisin varieties grown in Turkey were obtained with CVS. A total of 900 raisin grains were used, including 450 pieces from both varieties. These images were subjected to various stages of pre-processing and 7 morphological features were extracted.

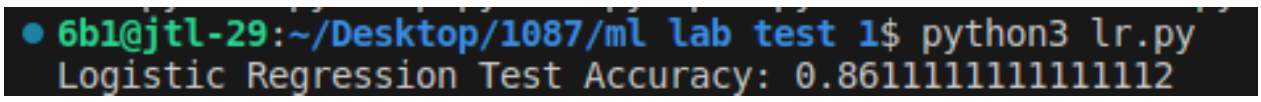
IMPLEMENTATION STEPS:

1) Load the dataset
2) Pre-process the data by handling missing values, encoding, normalization, standardization
3) Exploratory data analysis
4) Feature engineering techniques
5) Split the data into training, testing and validation sets
6) Train the model
7) Test the model
8) Measure the performance of the trained model
9) Represent the training and testing results using ROC curves.
10) Based on results obtained comment if the model overfits or not.

LINEAR REGRESSION MODEL:

```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])

scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.linear_model import LogisticRegression
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)

#test the model
lr_test_accuracy = lr_model.score(X_test, y_test)
print(f"Logistic Regression Test Accuracy: {lr_test_accuracy}")
```

OUTPUT:

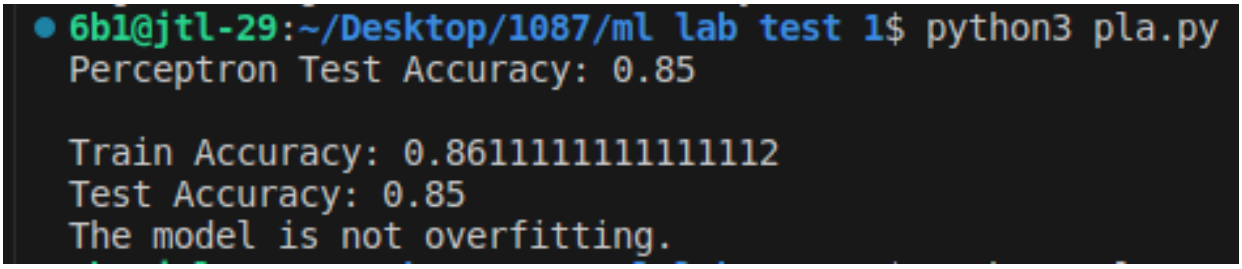


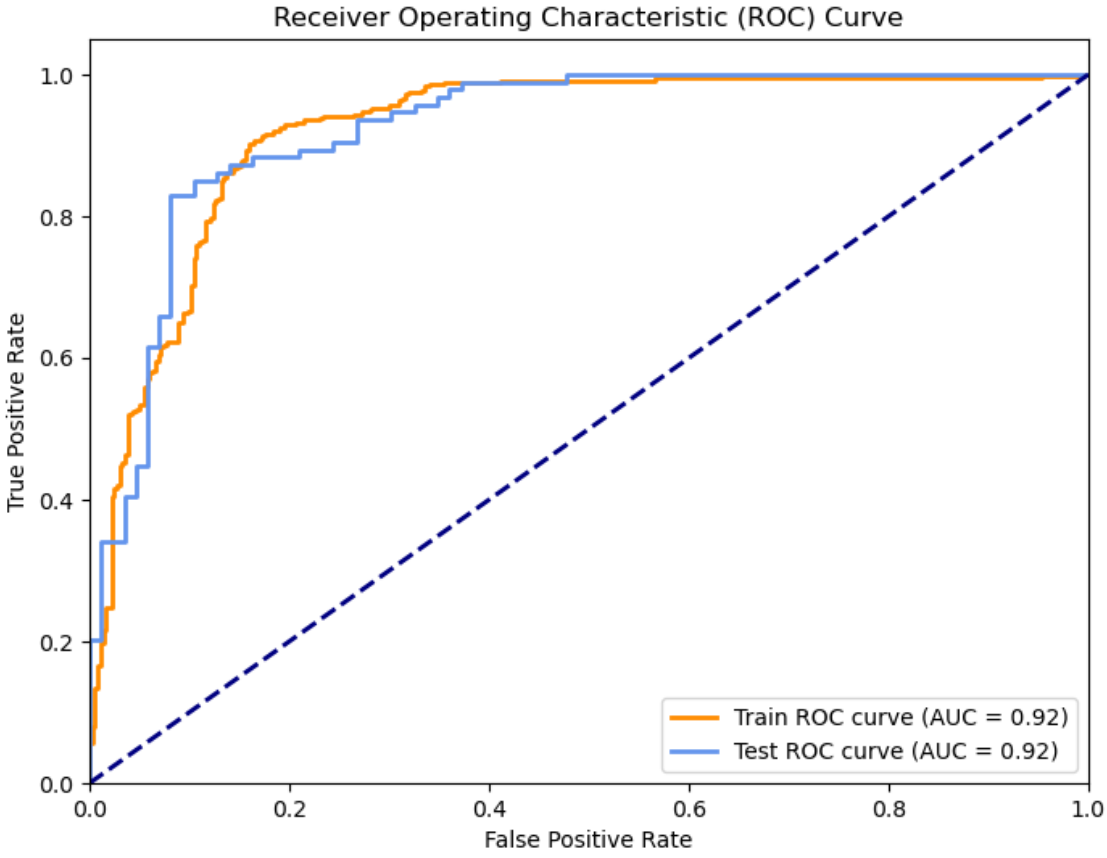
PERCEPTRON LEARNING ALGORITHM MODEL:

```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])

scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.linear_model import Perceptron
perceptron_model = Perceptron()
perceptron_model.fit(X_train, y_train)

#test the model
perceptron_test_accuracy = perceptron_model.score(X_test, y_test)
print(f"Perceptron Test Accuracy: {perceptron_test_accuracy}\n")


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#compute decision function scores
decision_train = perceptron_model.decision_function(X_train)
decision_test = perceptron_model.decision_function(X_test)

#compute roc curve and auc for training set
fpr_train, tpr_train, _ = roc_curve(y_train, decision_train)
roc_auc_train = auc(fpr_train, tpr_train)

#compute roc curve and auc for test set
```

```
fpr_test, tpr_test, _ = roc_curve(y_test, decision_test)
roc_auc_test = auc(fpr_test, tpr_test)

#plot roc curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='cornflowerblue', lw=2, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

#compute accuracy on training set
train_accuracy = perceptron_model.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")

#compute accuracy on test set
test_accuracy = perceptron_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Check for significant difference in accuracies
if train_accuracy - test_accuracy > 0.05:
print("The model might be overfitting.")
else:
print("The model is not overfitting.")
```

OUTPUT:



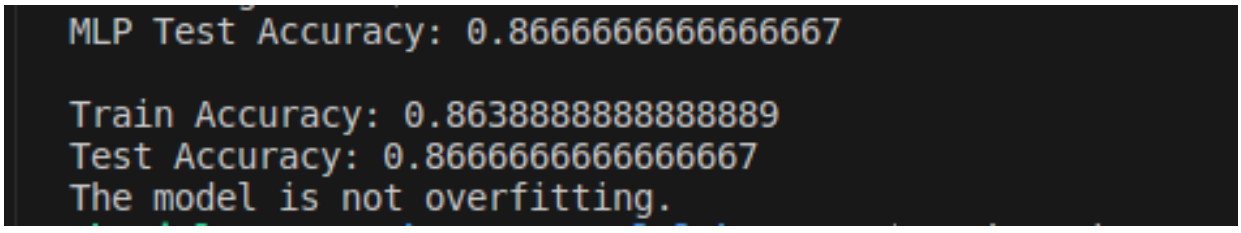

Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Train ROC curve (AUC = 0.92)
Test ROC curve (AUC = 0.92)

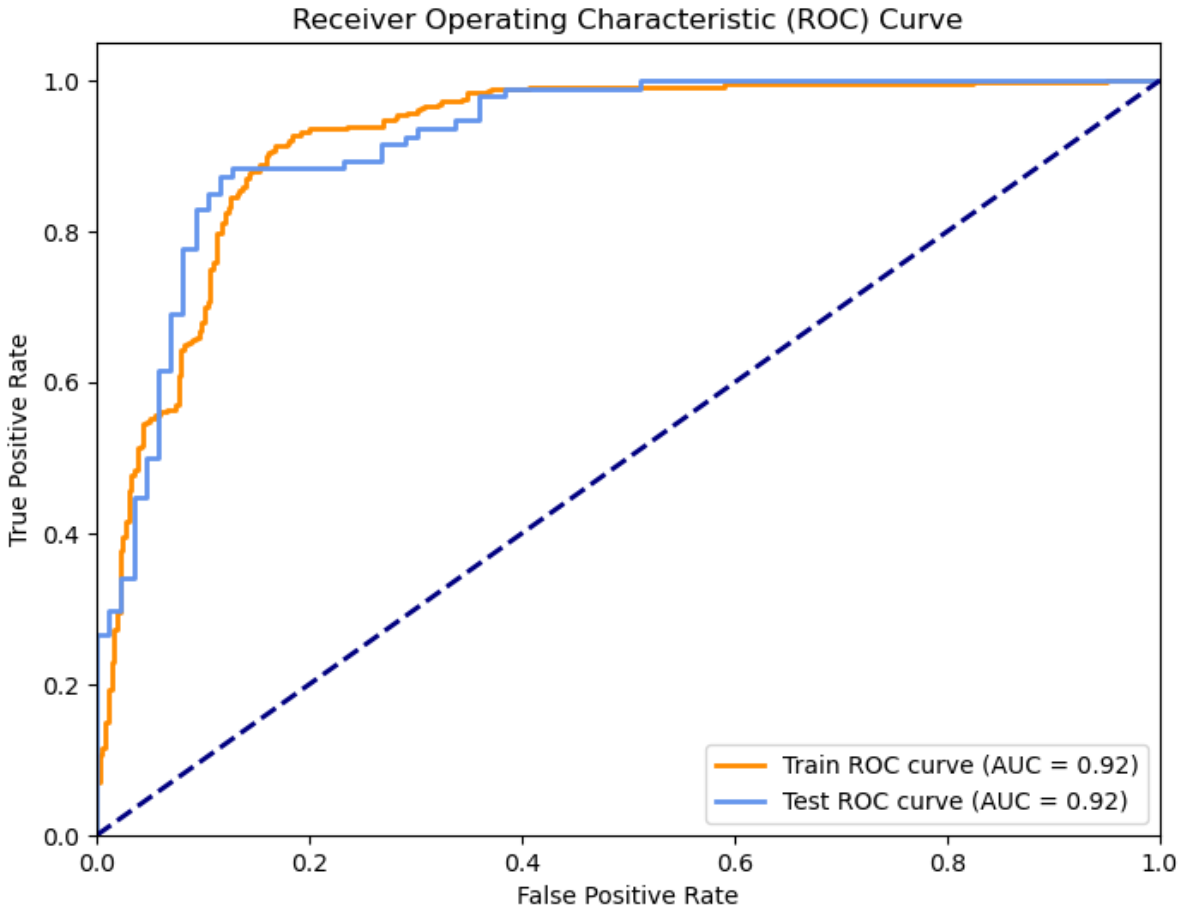
MULTI-LAYER PERCEPTRON MODEL:

```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])

scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.neural_network import MLPClassifier
mlp_model = MLPClassifier()
mlp_model.fit(X_train, y_train)

#test the model
mlp_test_accuracy = mlp_model.score(X_test, y_test)
print(f"MLP Test Accuracy: {mlp_test_accuracy}\n")


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#compute predictions
y_pred_train = mlp_model.predict_proba(X_train)[:, 1]
```

```
y_pred_test = mlp_model.predict_proba(X_test)[:, 1]

#compute roc curve and auc for training set
fpr_train, tpr_train, _ = roc_curve(y_train, y_pred_train)
roc_auc_train = auc(fpr_train, tpr_train)

#compute roc curve and auc for test set
fpr_test, tpr_test, _ = roc_curve(y_test, y_pred_test)
roc_auc_test = auc(fpr_test, tpr_test)

#plot roc curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='cornflowerblue', lw=2, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

#compute accuracy on training set
train_accuracy = mlp_model.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")

#compute accuracy on test set
test_accuracy = mlp_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Check for significant difference in accuracies
if train_accuracy - test_accuracy > 0.05:
print("The model might be overfitting.")
else:
print("The model is not overfitting.")
```

OUTPUT:



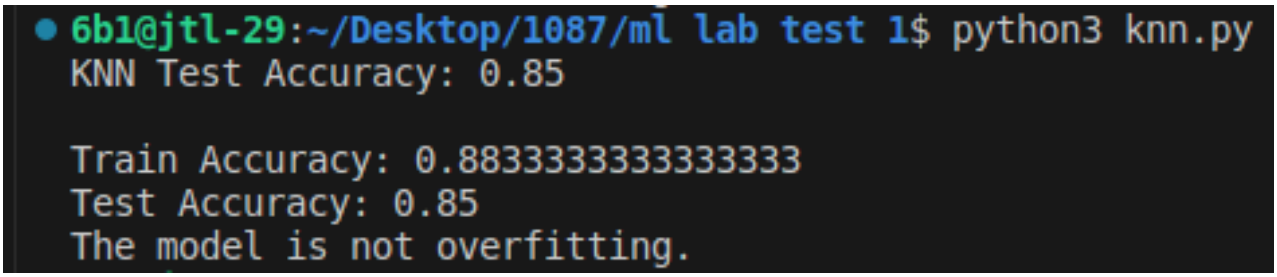

Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Train ROC curve (AUC = 0.92)
Test ROC curve (AUC = 0.92)

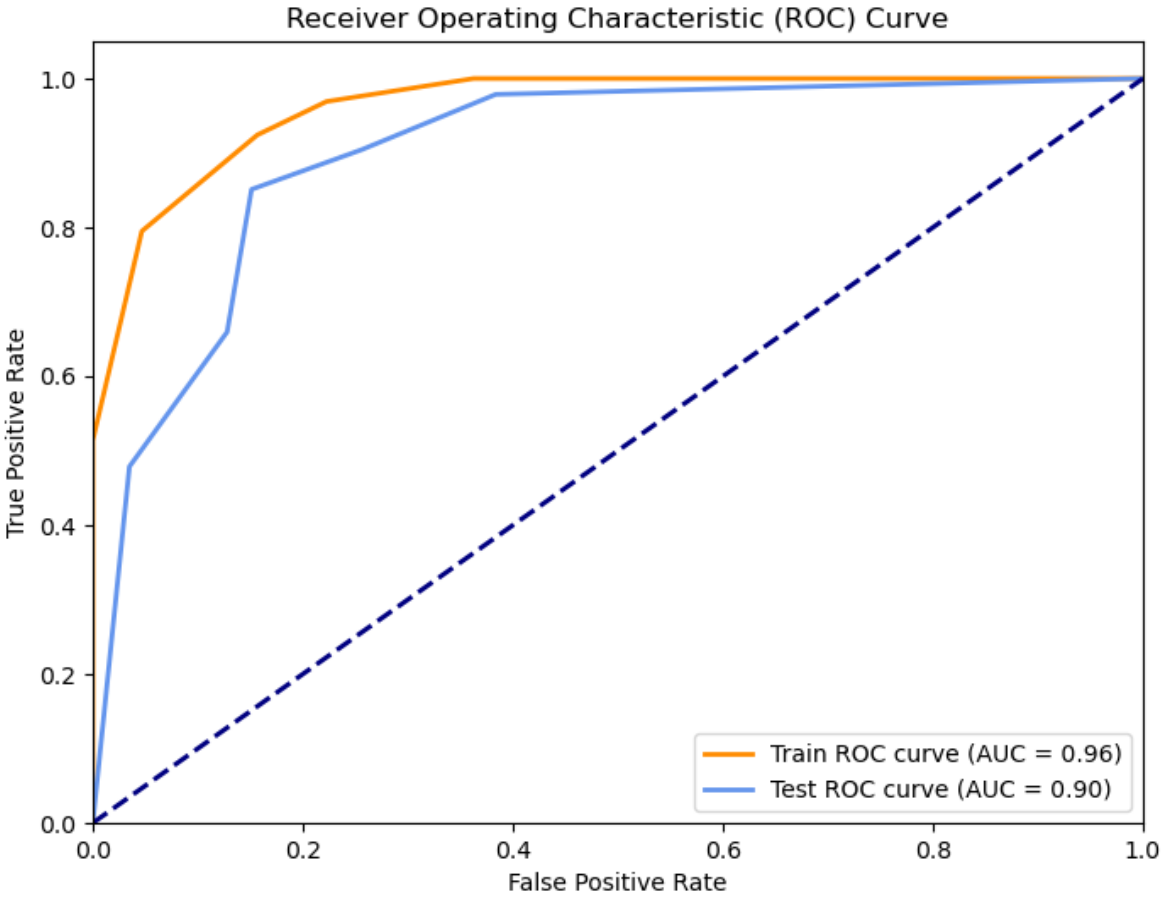
K-NEAREST NEIGHBORS MODEL:

```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])

scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier()
knn_model.fit(X_train, y_train)

#test the model
knn_test_accuracy = knn_model.score(X_test, y_test)
print(f"KNN Test Accuracy: {knn_test_accuracy}\n")


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#compute predictions
y_pred_train = knn_model.predict_proba(X_train)[:, 1]
```

```
y_pred_test = knn_model.predict_proba(X_test)[:, 1]

#compute roc curve and auc for training set
fpr_train, tpr_train, _ = roc_curve(y_train, y_pred_train)
roc_auc_train = auc(fpr_train, tpr_train)

#compute roc curve and auc for test set
fpr_test, tpr_test, _ = roc_curve(y_test, y_pred_test)
roc_auc_test = auc(fpr_test, tpr_test)

#plot roc curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='cornflowerblue', lw=2, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

#compute accuracy on training set
train_accuracy = knn_model.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")

#compute accuracy on test set
test_accuracy = knn_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Check for significant difference in accuracies
if train_accuracy - test_accuracy > 0.05:
print("The model might be overfitting.")
else:
print("The model is not overfitting.")
```

OUTPUT:




Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
Train ROC curve (AUC = 0.96)
Test ROC curve (AUC = 0.90)

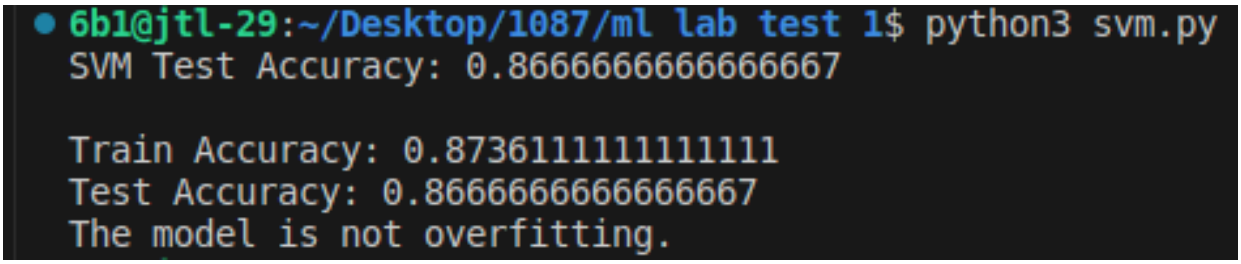
SUPPORT VECTOR MACHINE MODEL:

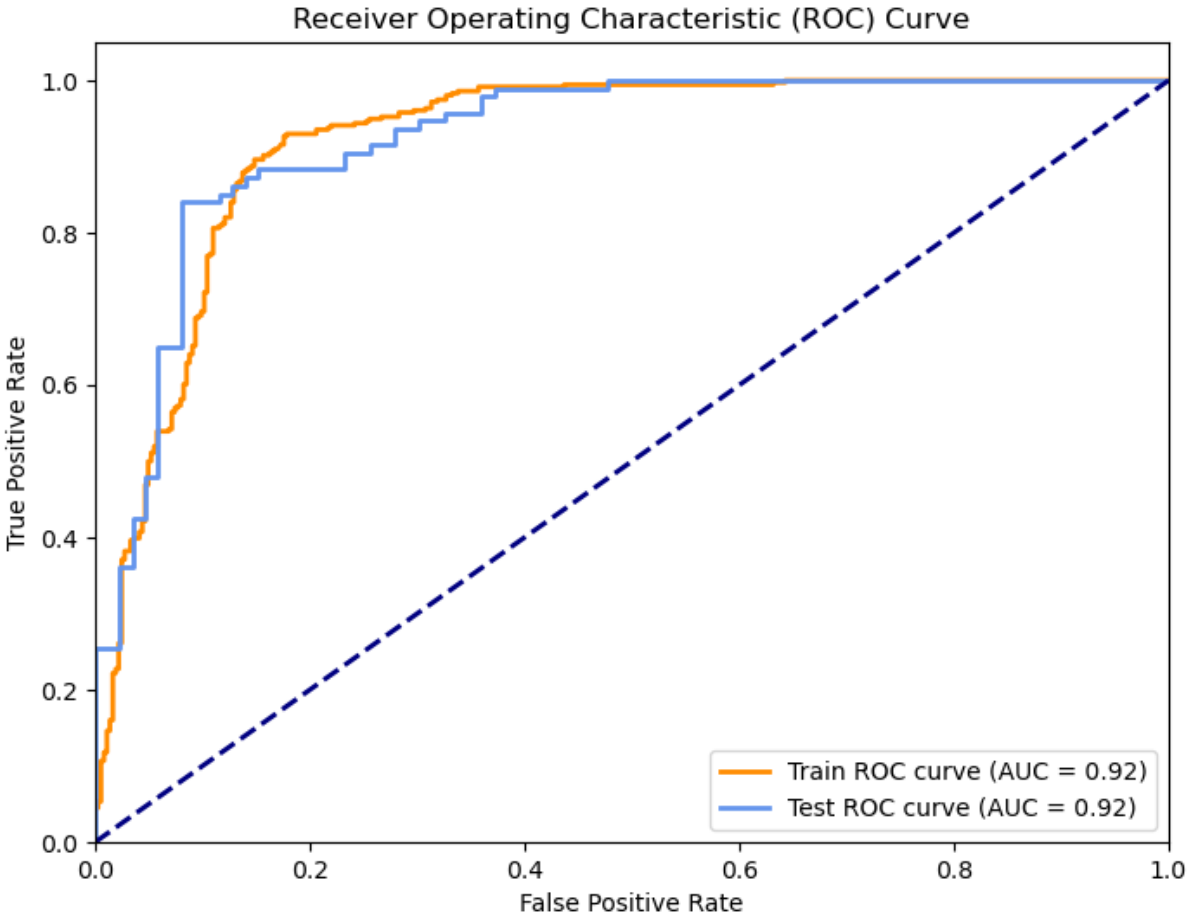
```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])
```

```
scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.svm import SVC
svm_model = SVC(probability=True) # Enabling probability estimation for ROC curve
svm_model.fit(X_train, y_train)

#test the model
svm_test_accuracy = svm_model.score(X_test, y_test)
print(f"SVM Test Accuracy: {svm_test_accuracy}\n")


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#compute predictions
y_pred_train = svm_model.predict_proba(X_train)[:, 1]
y_pred_test = svm_model.predict_proba(X_test)[:, 1]

#compute roc curve and auc for training set
fpr_train, tpr_train, _ = roc_curve(y_train, y_pred_train)
roc_auc_train = auc(fpr_train, tpr_train)

#compute roc curve and auc for test set
fpr_test, tpr_test, _ = roc_curve(y_test, y_pred_test)
roc_auc_test = auc(fpr_test, tpr_test)

#plot roc curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='cornflowerblue', lw=2, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

#compute accuracy on training set
train_accuracy = svm_model.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")
```

```
#compute accuracy on test set
test_accuracy = svm_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Check for significant difference in accuracies
if train_accuracy - test_accuracy > 0.05:
print("The model might be overfitting.")
else:
print("The model is not overfitting.")
```

OUTPUT:



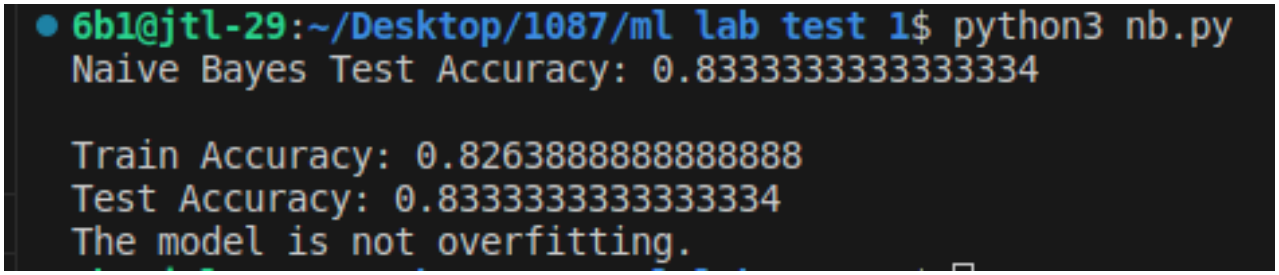


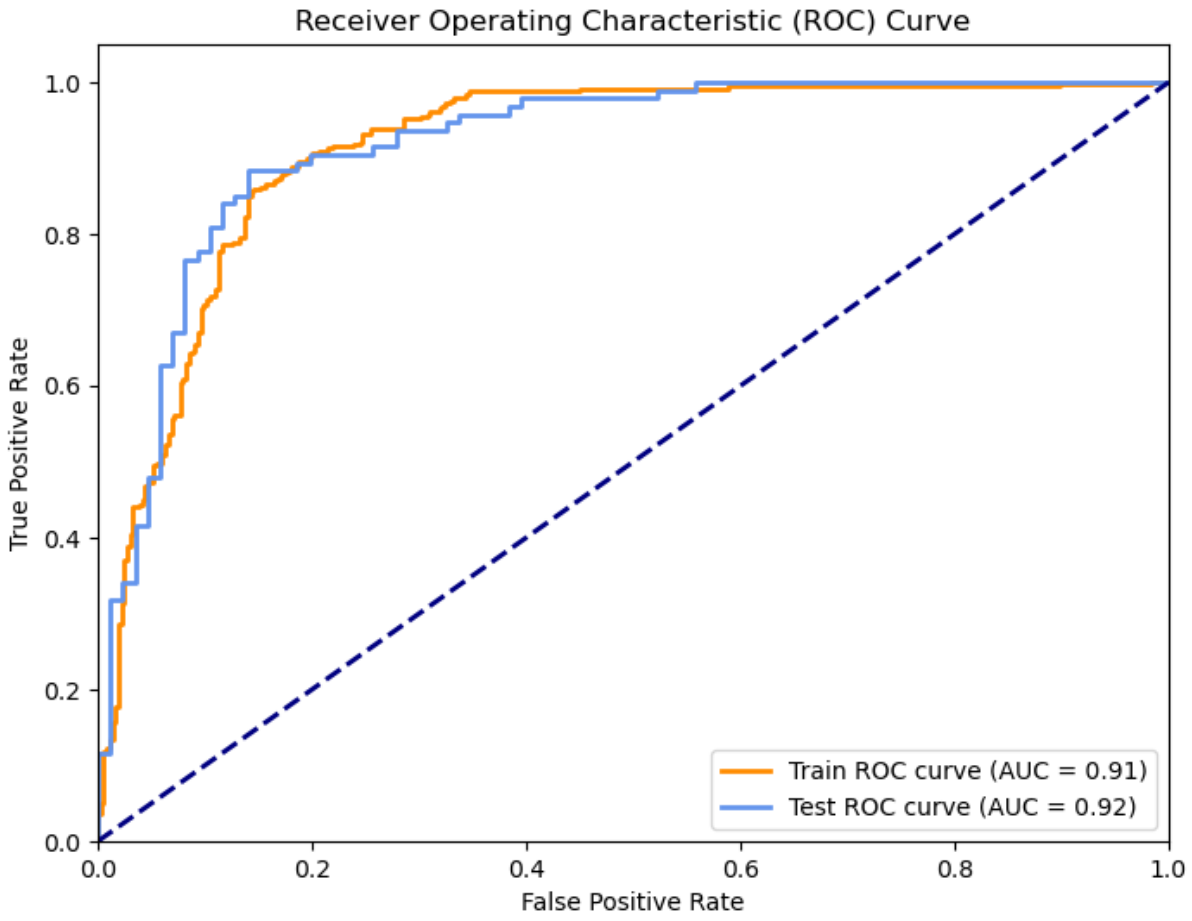
NAIVE BAYES MODEL:

```
#load dataset
import pandas as pd
data = pd.read_csv("raisin.csv")

#preprocess data
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler
from sklearn.impute import SimpleImputer

label_encoder = LabelEncoder()
data['Class'] = label_encoder.fit_transform(data['Class'])

imputer = SimpleImputer(strategy='mean')
data.fillna(data.mean(), inplace=True)

scaler = StandardScaler()
data_scaled = scaler.fit_transform(data.iloc[:, :-1])

scaler = MinMaxScaler()
data_normalized = scaler.fit_transform(data_scaled)

#split data into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_normalized[:, :-1], data['Class'], test_size=0.2,
random_state=42)

#train the model
from sklearn.naive_bayes import GaussianNB
nb_model = GaussianNB()
nb_model.fit(X_train, y_train)

#test the model
nb_test_accuracy = nb_model.score(X_test, y_test)
print(f"Naive Bayes Test Accuracy: {nb_test_accuracy}\n")


from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

#compute predictions
y_pred_train = nb_model.predict_proba(X_train)[:, 1]
y_pred_test = nb_model.predict_proba(X_test)[:, 1]

#compute roc curve and auc for training set
fpr_train, tpr_train, _ = roc_curve(y_train, y_pred_train)
roc_auc_train = auc(fpr_train, tpr_train)

#compute roc curve and auc for test set
fpr_test, tpr_test, _ = roc_curve(y_test, y_pred_test)
```

```
roc_auc_test = auc(fpr_test, tpr_test)

#plot roc curves
plt.figure(figsize=(8, 6))
plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label=f'Train ROC curve (AUC = {roc_auc_train:.2f})')
plt.plot(fpr_test, tpr_test, color='cornflowerblue', lw=2, label=f'Test ROC curve (AUC = {roc_auc_test:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

#compute accuracy on training set
train_accuracy = nb_model.score(X_train, y_train)
print(f"Train Accuracy: {train_accuracy}")

#compute accuracy on test set
test_accuracy = nb_model.score(X_test, y_test)
print(f"Test Accuracy: {test_accuracy}")

# Check for significant difference in accuracies
if train_accuracy - test_accuracy > 0.05:
print("The model might be overfitting.")
else:
print("The model is not overfitting.")
```

OUTPUT:





LEARNING OUTCOME:

From all the models computed SVM has the best accuracy.