In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
```

In [ ]:
```
!gdown 1XI_gYaj-G3AOz-96j0PjCb9TcBFA6s49
```

```
Downloading...
From: https://drive.google.com/uc?id=1XI_gYaj-G3AOz-96j0PjCb9TcBFA6s49
To: /content/wdbc.data
100% 124k/124k [00:00<00:00, 90.2MB/s]
```

In [ ]:
```
cols = [
    'ID', 'diagnosis', 'radius1', 'texture1', 'perimeter1', 'area1', 'smoothness1',
    'compactness1', 'concavity1', 'concave_points1', 'symmetry1', 'fractal_dimension1',
    'radius2', 'texture2', 'perimeter2', 'area2', 'smoothness2', 'compactness2',
    'concavity2', 'concave_points2', 'symmetry2', 'fractal_dimension2', 'radius3',
    'texture3', 'perimeter3', 'area3', 'smoothness3', 'compactness3', 'concavity3',
    'concave_points3', 'symmetry3', 'fractal_dimension3'
]
df = pd.read_csv("wdbc.data", names=cols)
```

In [ ]:
```
X = df.drop(['ID', 'diagnosis'], axis=1)
y = df['diagnosis']
y = np.where(y == 'M', 0, 1)
```

In [ ]:
```
# Function for splitting, scaling, and sampling the dataset
def preprocessing(X, y, test_split, scaler, sampler):
  if(scaler != None):
    X = scaler.fit_transform(X)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_split, random_state=10)
  if(sampler != None):
    X_train, y_train = RandomOverSampler().fit_resample(X_train, y_train)
  return X_train, y_train, X_test, y_test
```

In [ ]:

```
# Function for training and prediction
def trainAndPredict(model, X_train, y_train, X_test, y_test):
  model = model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  report = classification_report(y_test, y_pred)
  conf_matrix = confusion_matrix(y_test, y_pred)
  return accuracy, report, conf_matrix
```

In [ ]:

```
# Function for generating an image illustrating Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC)
def generateROCAndAUC(X, y, test_size, model, scaler, sampler):
  X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=scaler, sampler=sampler, test_split=test_size)
  model.fit(X_train, y_train)
  if hasattr(model, "predict_proba"):
      probas = model.predict_proba(X_test)
      prob_positive_class = probas[:, 1]
  else:
      decision_function = model.decision_function(X_test)
      prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

  fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
  roc_auc = auc(fpr, tpr)

  plt.figure(figsize=(8, 6))
  plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
  plt.plot([0, 1], [0, 1], color='gray', 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(f'ROC Curve for Binary Classification')
  plt.legend(loc='lower right')
  plt.show()
```

In [ ]:

```
def performOperation(X, y, model, title, scaler=None, sampler=None):
  print("\n\n**********************************************************************************************************************
  print(title);
  graph_x = []
  graph_y = []
  max_accuracy = 0
  test_size = 0
  final_report = None
  final_conf_matrix = None
  for test_split in range(3, 8):
```

```
    X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=scaler, sampler=sampler,  test_split=test_split*0.1)
    accuracy, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
    if max_accuracy < accuracy:
      test_size = test_split*0.1
      max_accuracy = accuracy
      final_report = report
      final_conf_matrix = conf_matrix
    graph_x.append(test_split*10)
    graph_y.append(accuracy*100)
  print("Got Maximum Accuracy for Test Size = ", int(test_size*100), "%")
  print("Maximum Accuracy = ", max_accuracy*100, "%")
  print("Performance Evolution:")
  print(final_report)
  print("Confusion Matrix:")
  sns.heatmap(final_conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
  plt.plot(graph_x, graph_y, marker='o')
  plt.xlabel('Test Size')
  plt.ylabel('Accuracy')
  plt.title(title)
  plt.show()
  print("\n")
  generateROCAndAUC(X, y, test_size=test_size, model=model, scaler=scaler, sampler=sampler)
  print("\n\n**********************************************************************************************************************************")
  return graph_y
```

In [ ]:

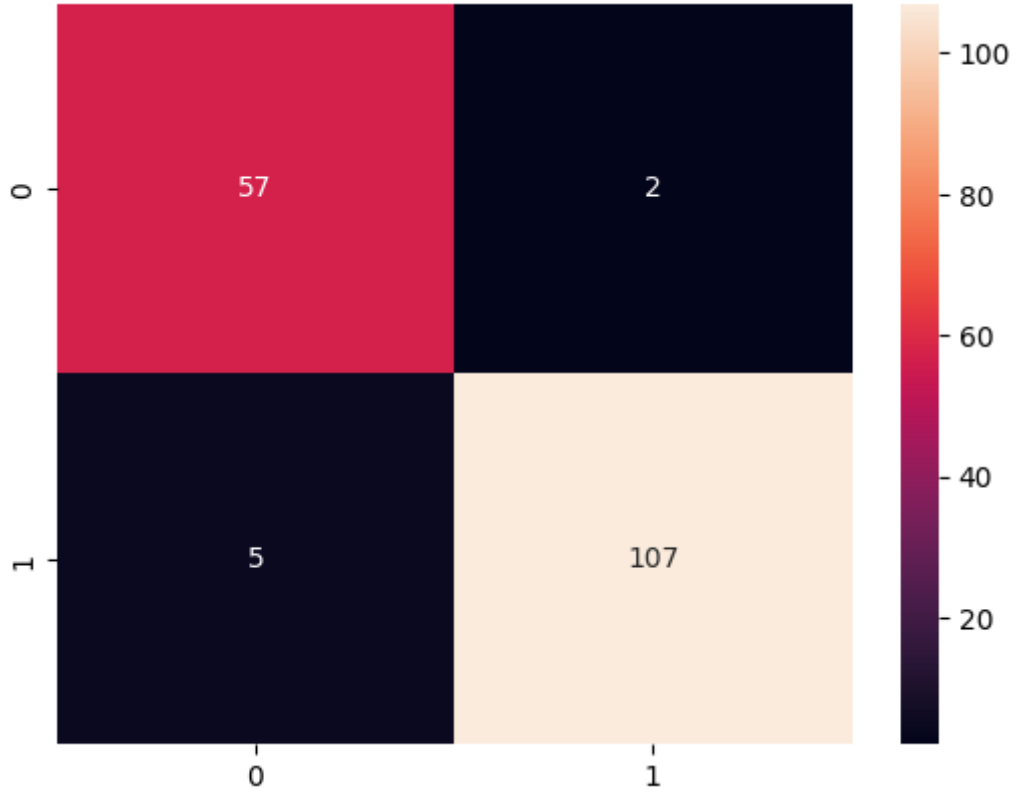
```
def performOperationPCA(X, y, model, test_split, sampler=None, scaler=None):
  X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=None, sampler=sampler,  test_split=test_split)
  _, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
  print("Performance Evolution:")
  print(report)
  print("Confusion Matrix:")
  sns.heatmap(conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
```

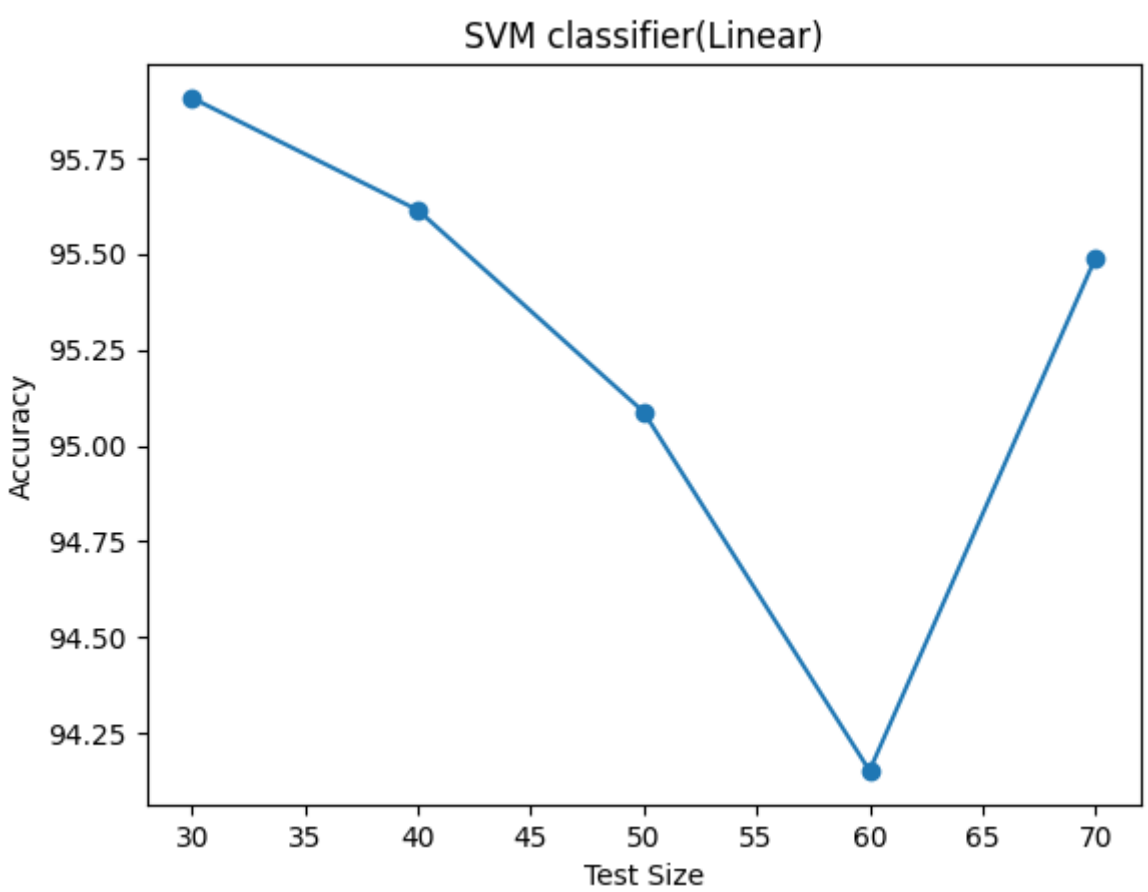
In [ ]:

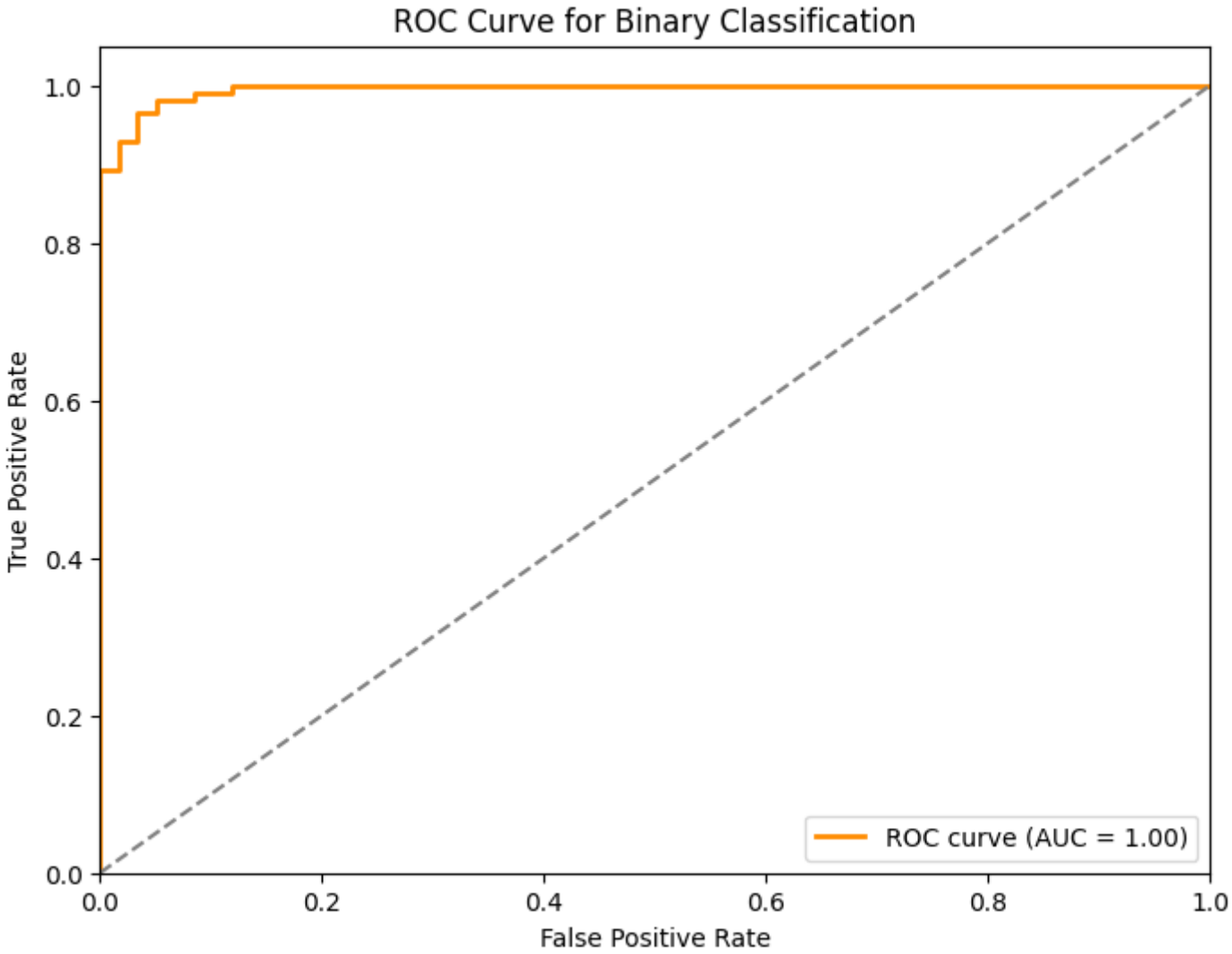
```
svm_data = []
svm_data.append(performOperation(X, y, model=SVC(kernel='linear'), title="SVM classifier(Linear)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='poly'), title="SVM classifier(Polynomial)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='sigmoid', gamma=0.01), title="SVM classifier(Sigmoid)", sampler=RandomOverSampler(),
svm_data.append(performOperation(X, y, model=SVC(kernel='rbf'), title="SVM classifier(Gaussian)", sampler=RandomOverSampler()))
```

****

```
*********************************************************


SVM classifier(Linear)
Got Maximum Accuracy for Test Size =  30 %
Maximum Accuracy =  95.90643274853801 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.92      0.97      0.94        59
           1       0.98      0.96      0.97       112

    accuracy                           0.96       171
   macro avg       0.95      0.96      0.96       171
weighted avg       0.96      0.96      0.96       171

Confusion Matrix:
```




SVM classifier(Linear)
Accuracy
95.75
95.50
95.25
95.00
94.75
94.50
94.25
30
35
40
45
50
55
60
65
70
Test Size



****

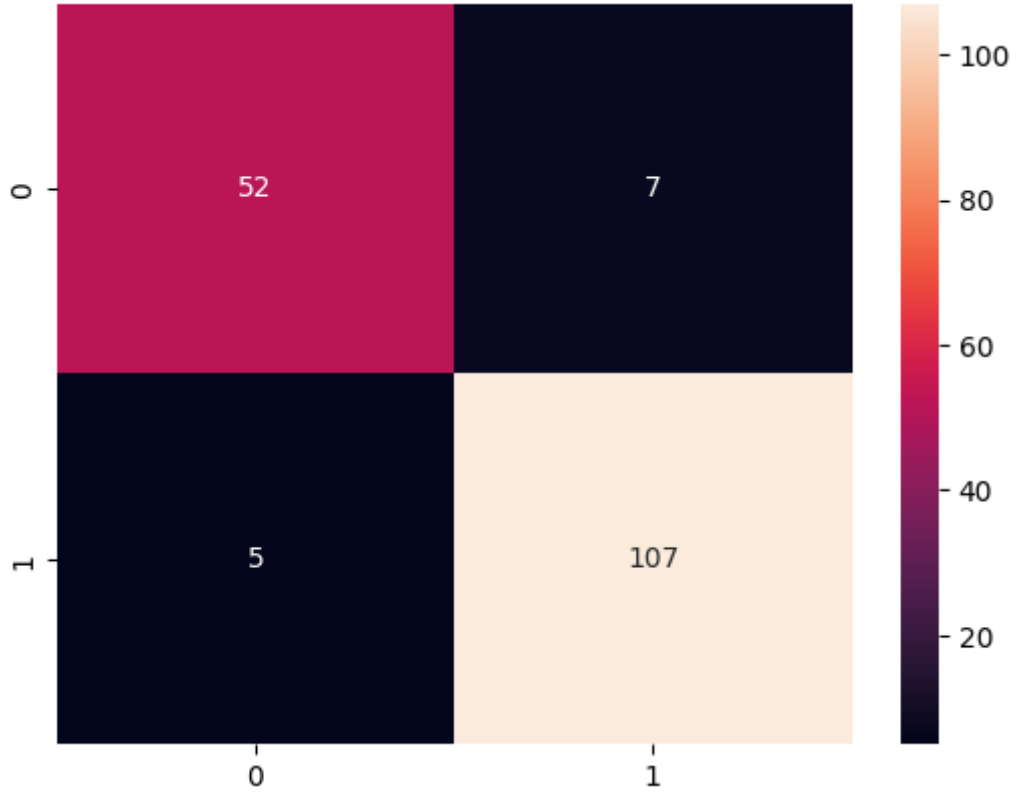
****

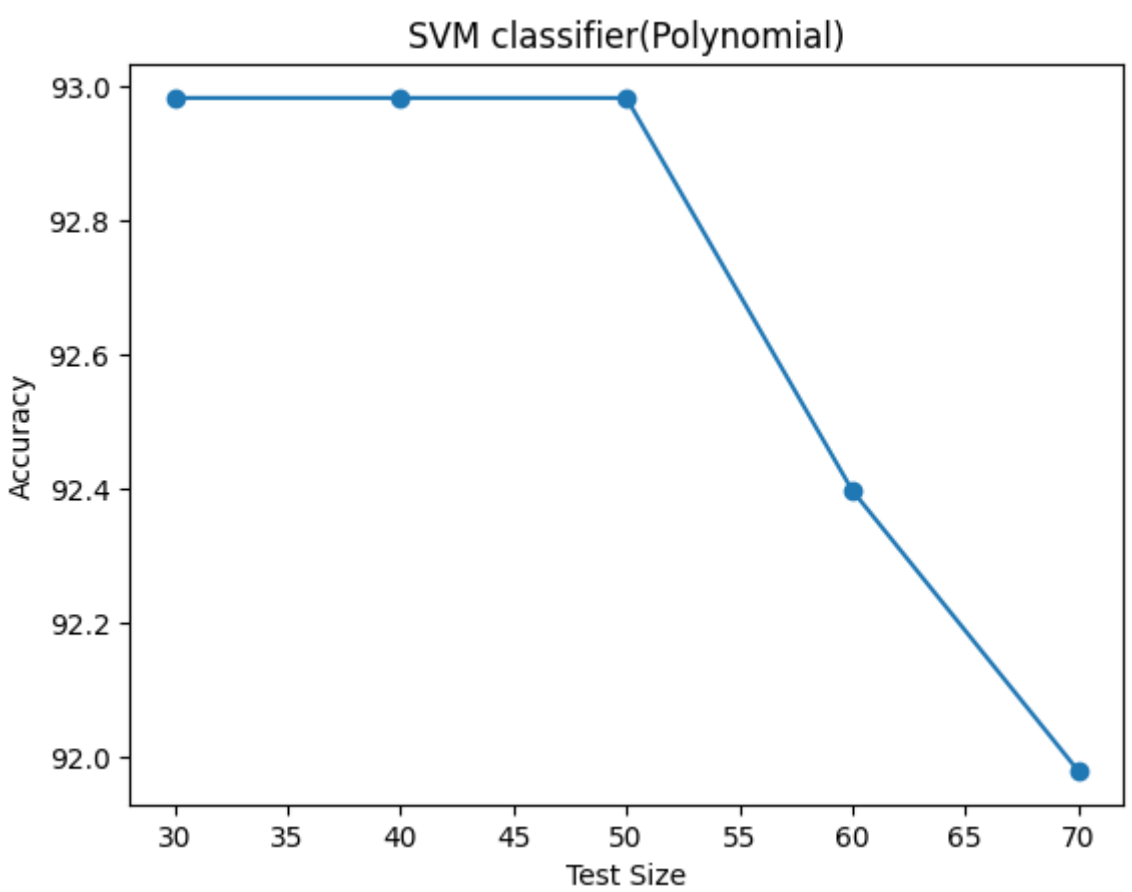
SVM classifier(Polynomial)
Got Maximum Accuracy for Test Size = 30 %
Maximum Accuracy = 92.98245614035088 %

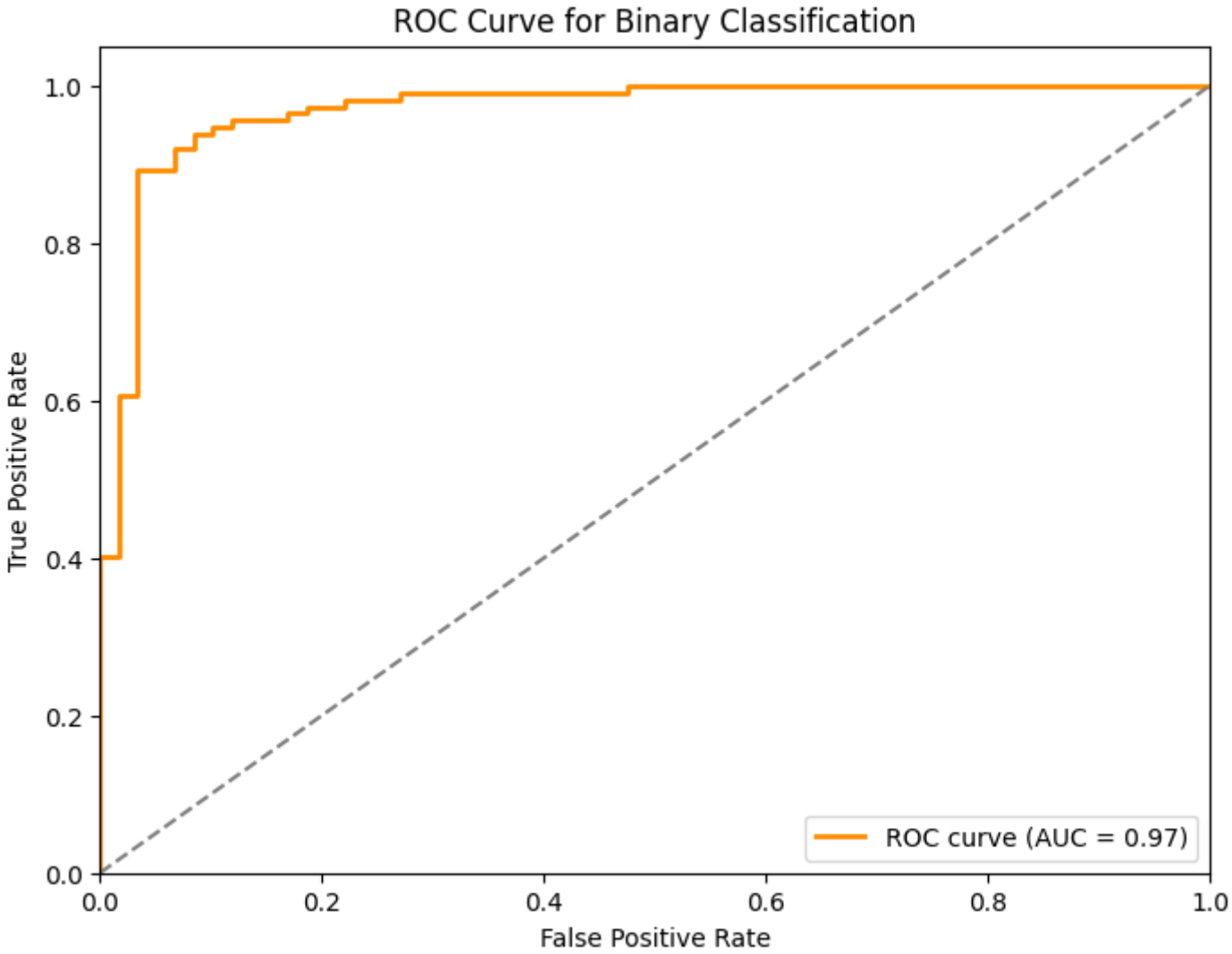
Performance Evolution:

	precision	recall	f1-score	support
0	0.91	0.88	0.90	59
1	0.94	0.96	0.95	112
accuracy			0.93	171
macro avg	0.93	0.92	0.92	171
weighted avg	0.93	0.93	0.93	171

Confusion Matrix:




SVM classifier(Polynomial)
93.0
92.8
92.6
92.4
92.2
92.0
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

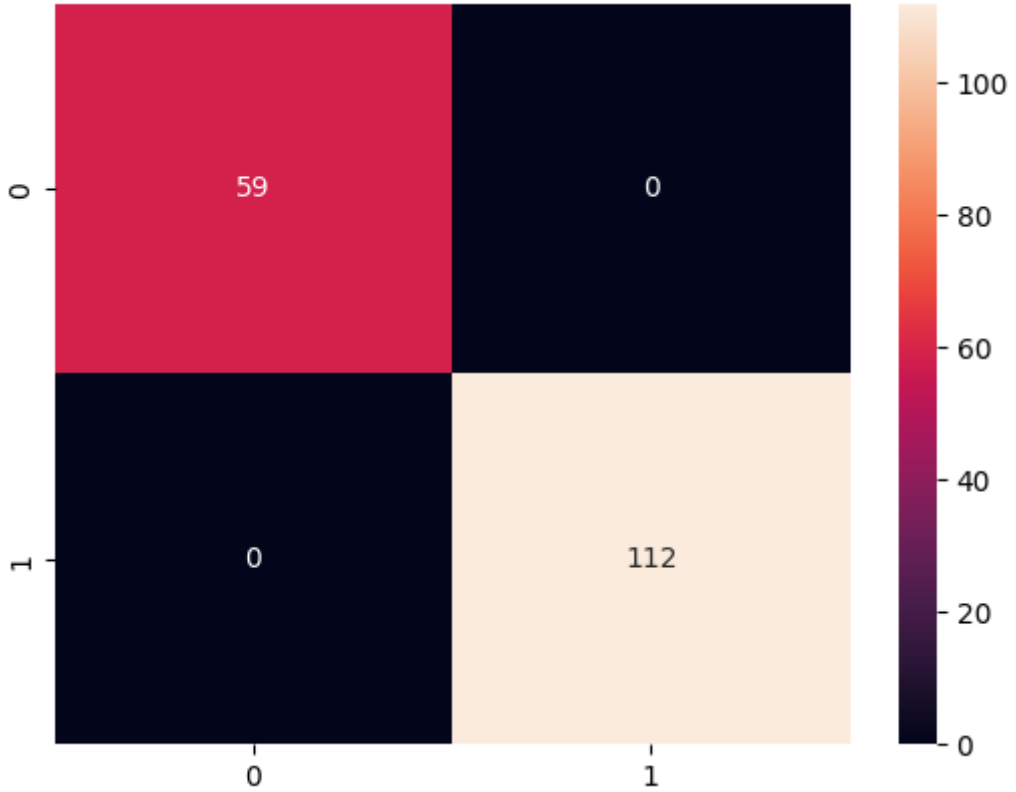
****

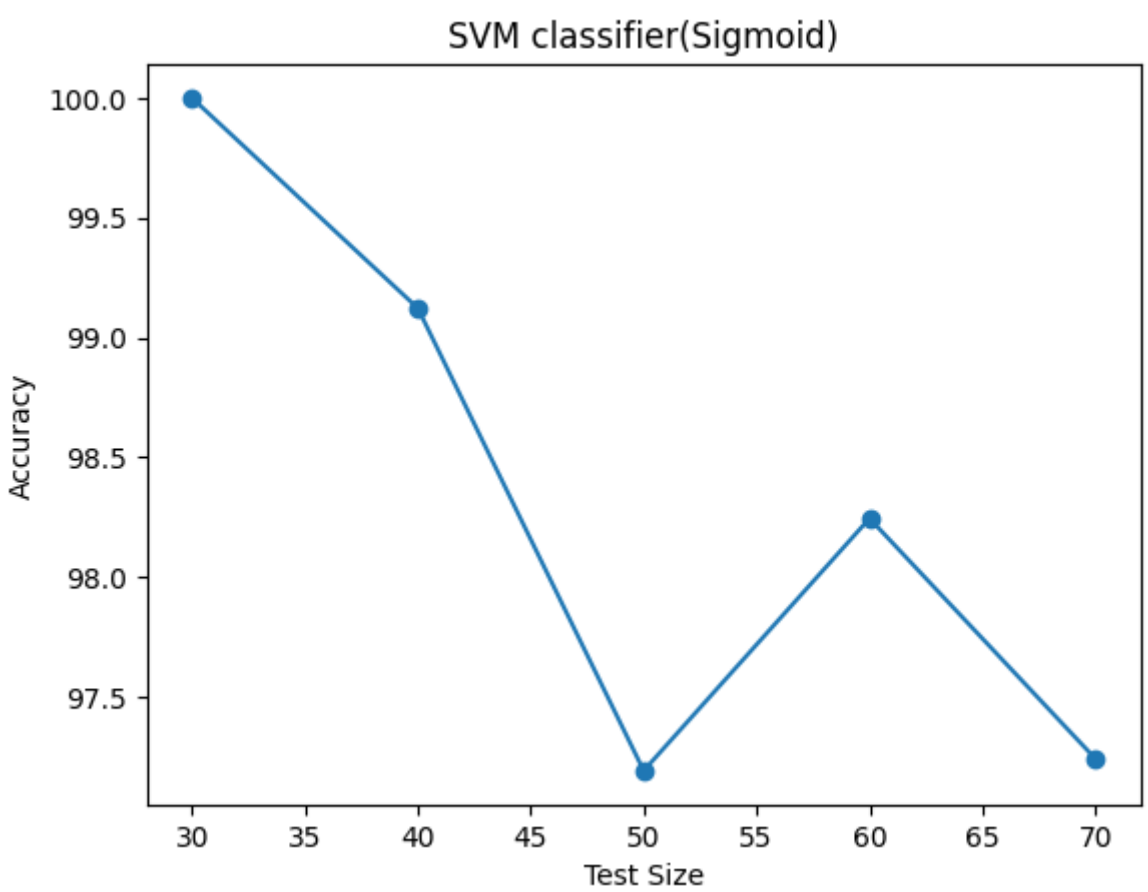
SVM classifier(Sigmoid)
Got Maximum Accuracy for Test Size = 30 %
Maximum Accuracy = 100.0 %

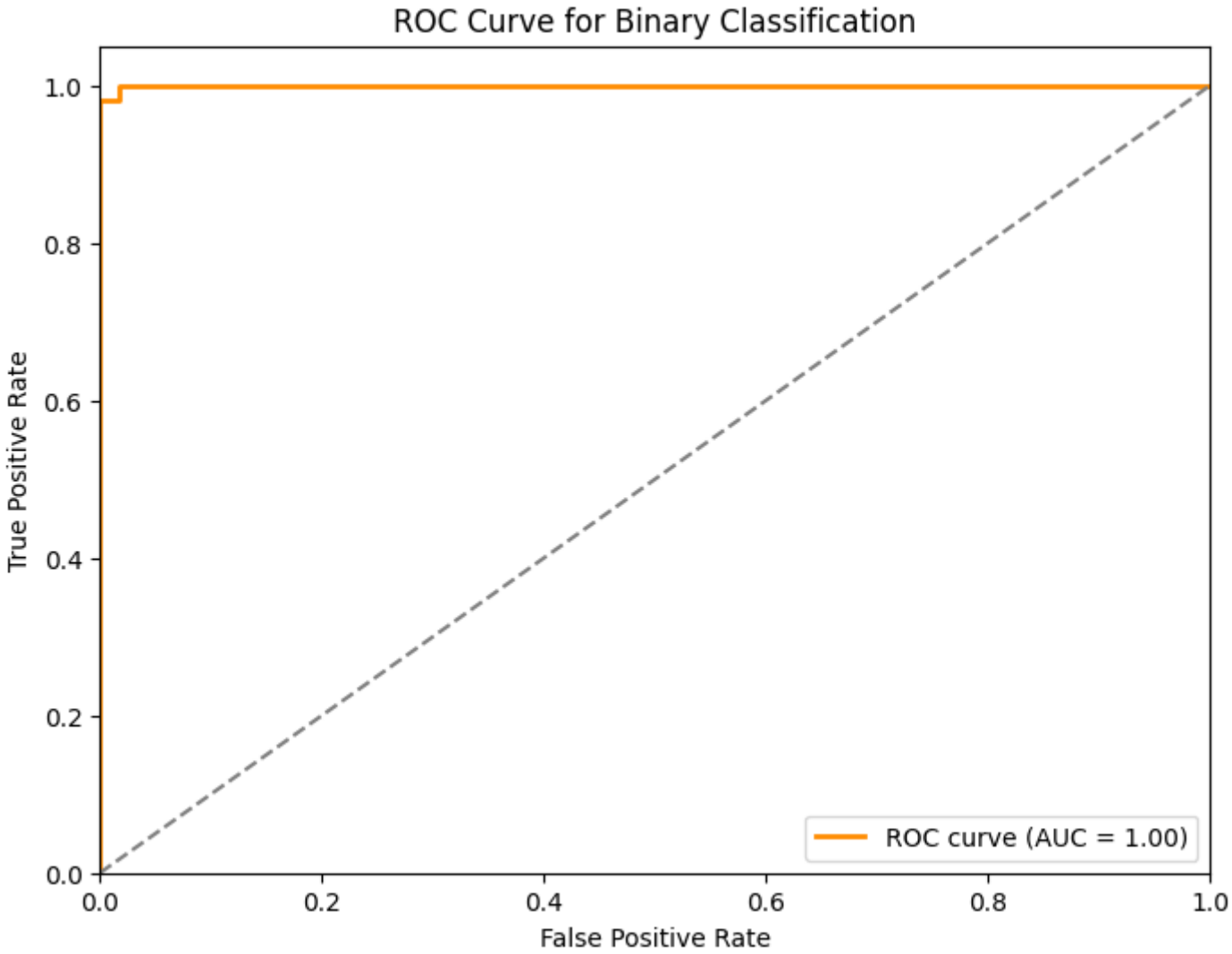
Performance Evolution:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	59
1	1.00	1.00	1.00	112
accuracy			1.00	171
macro avg	1.00	1.00	1.00	171
weighted avg	1.00	1.00	1.00	171

Confusion Matrix:




SVM classifier(Sigmoid)
100.0
99.5
99.0
98.5
98.0
97.5
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

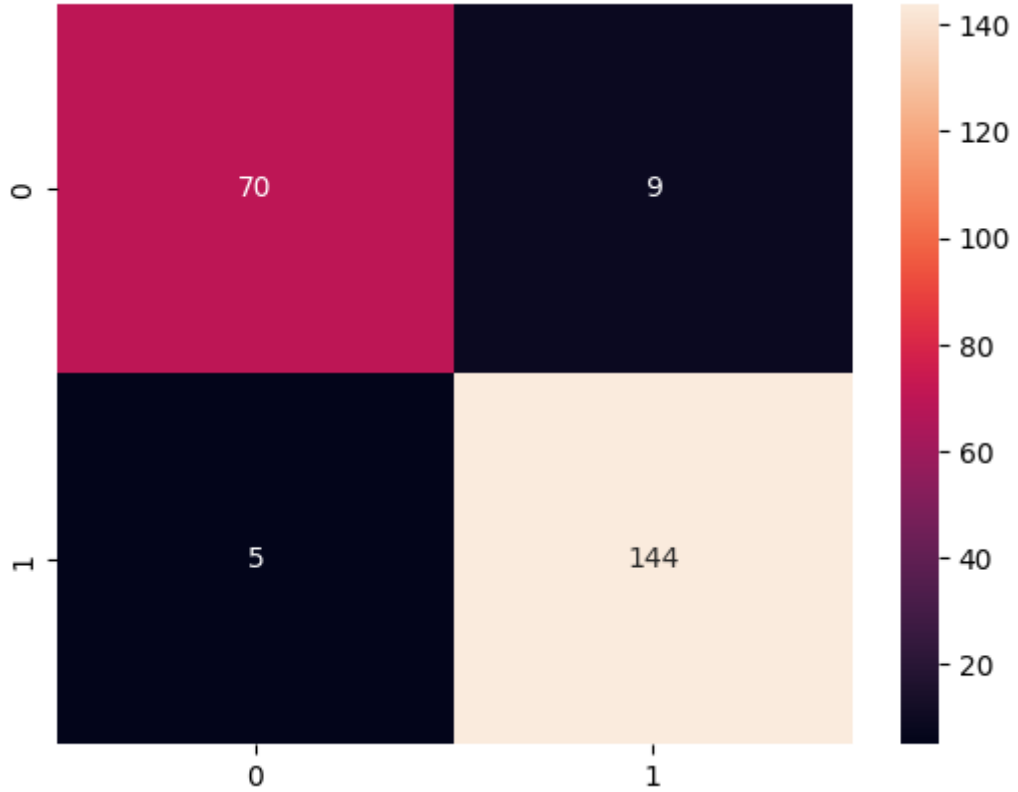
****

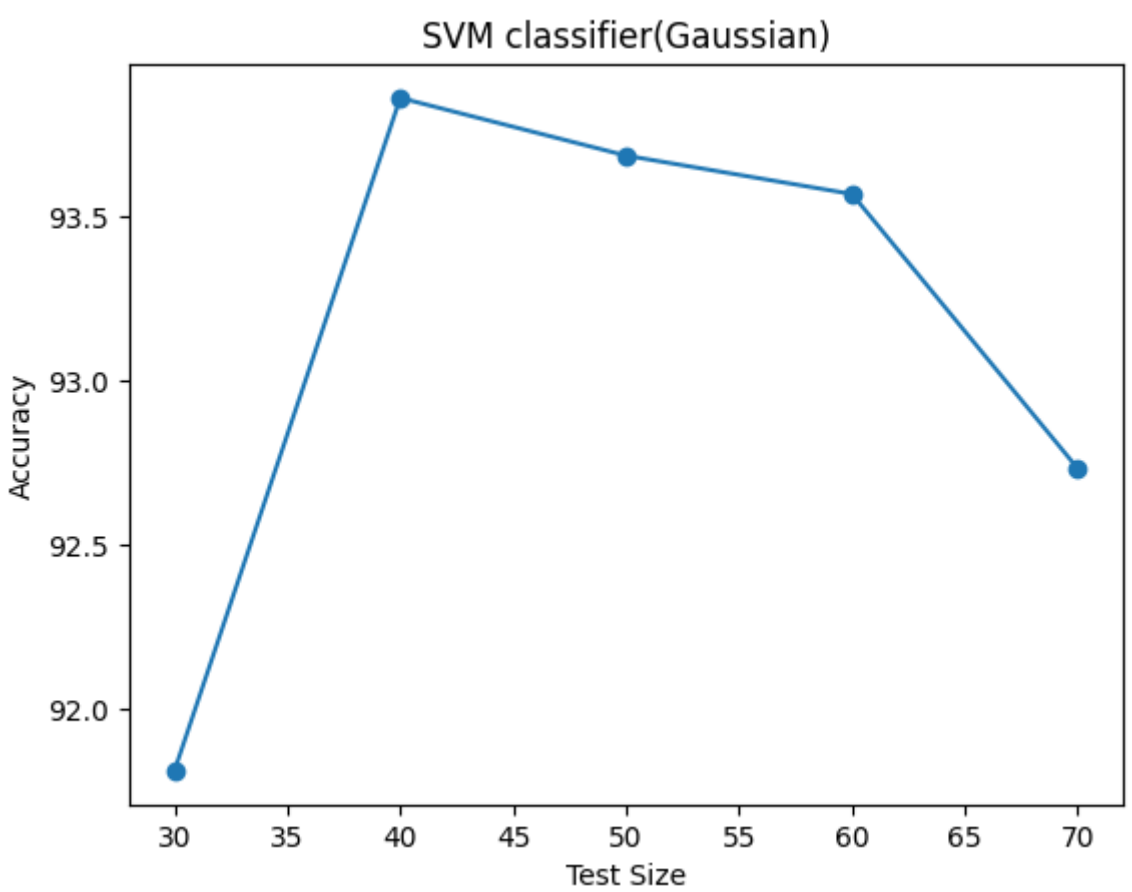
SVM classifier(Gaussian)
Got Maximum Accuracy for Test Size = 40 %
Maximum Accuracy = 93.85964912280701 %

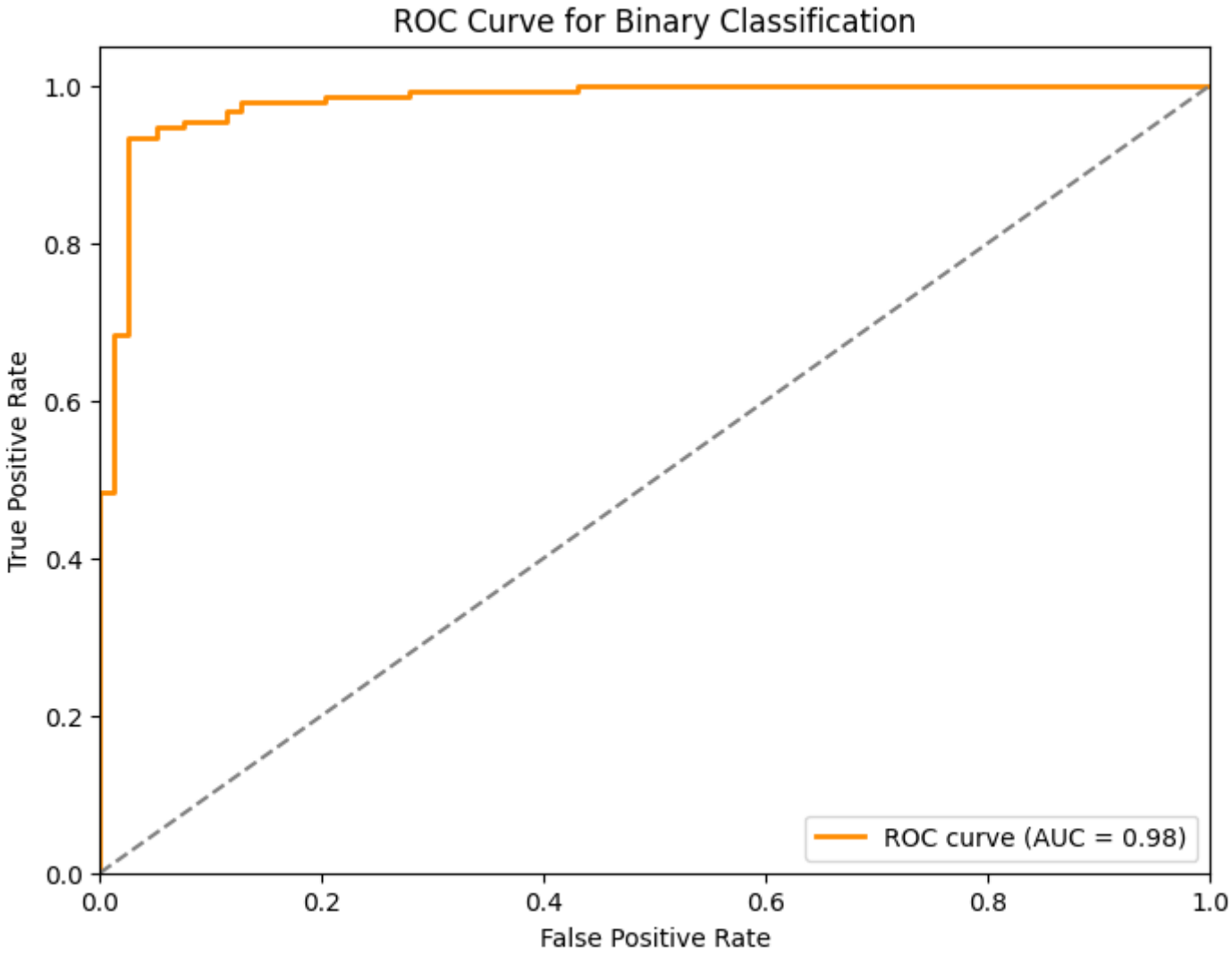
Performance Evolution:

	precision	recall	f1-score	support
0	0.93	0.89	0.91	79
1	0.94	0.97	0.95	149
accuracy			0.94	228
macro avg	0.94	0.93	0.93	228
weighted avg	0.94	0.94	0.94	228

Confusion Matrix:




SVM classifier(Gaussian)
Accuracy
93.5
93.0
92.5
92.0
30
35
40
45
50
55
60
65
70
Test Size



****

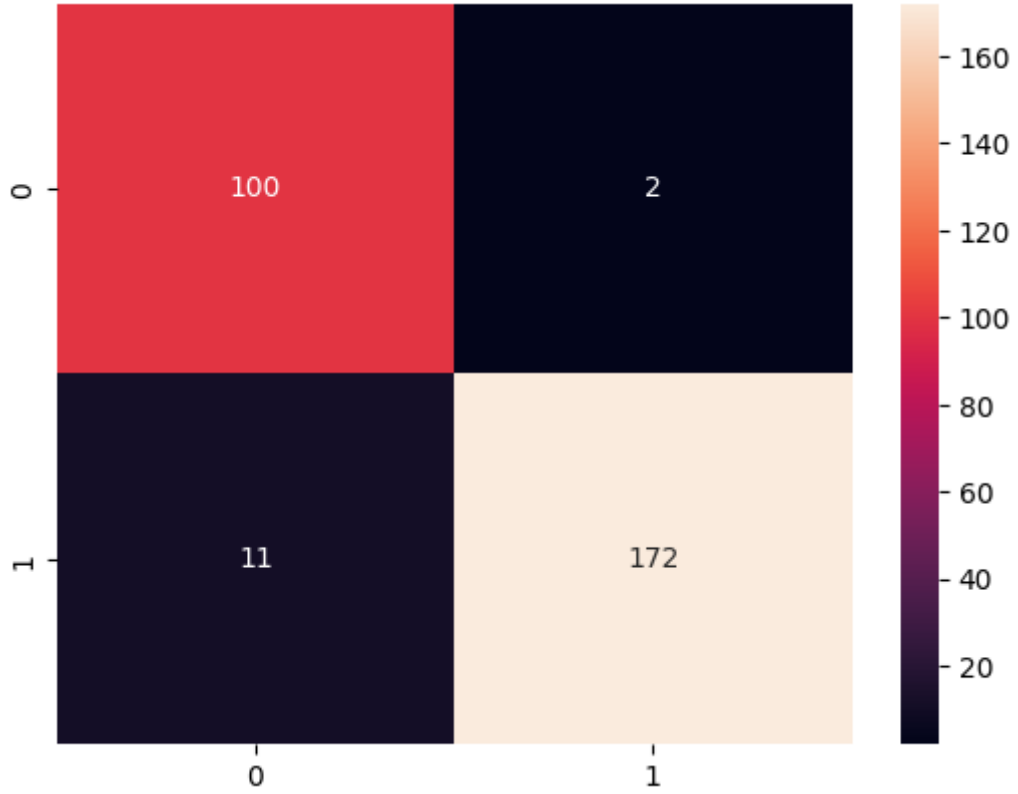
In [ ]:
```
_ = performOperation(X, y, model=MLPClassifier(max_iter=1000, learning_rate='adaptive'), title="MLP Classifier", sampler=RandomOverSampler())
```

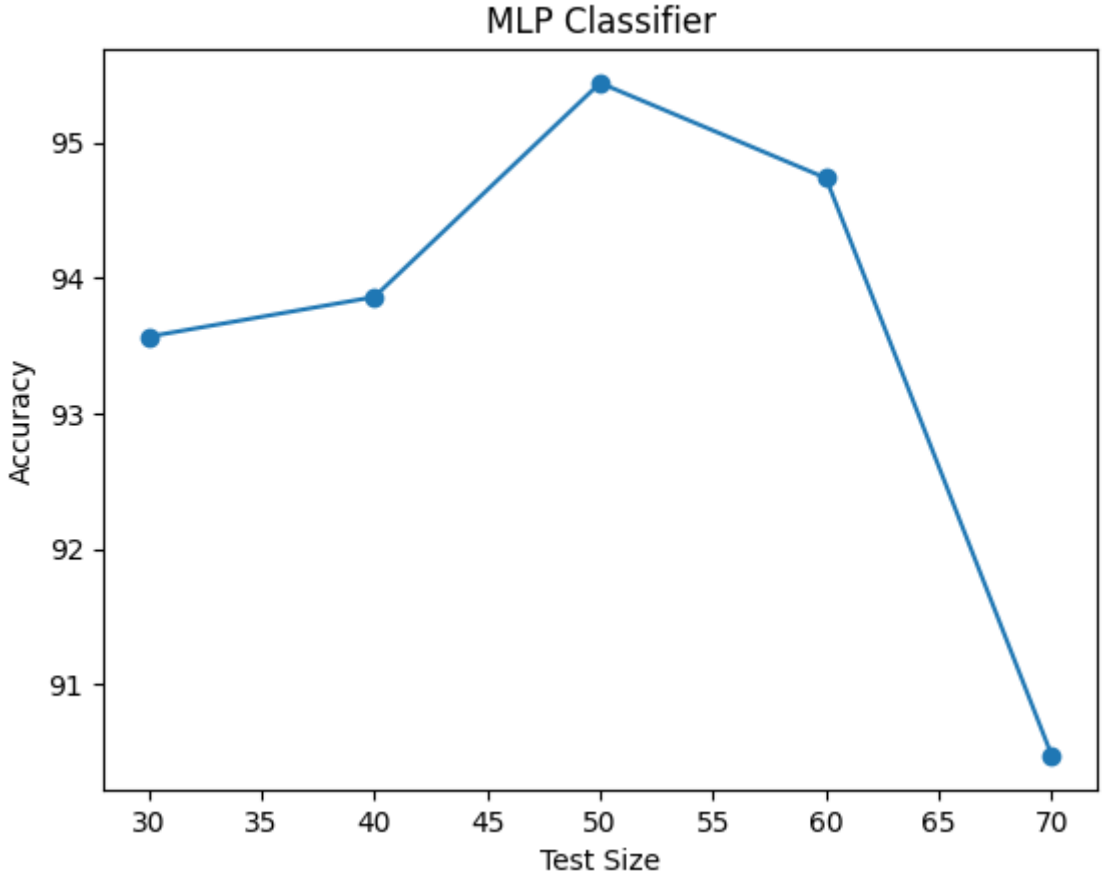
****

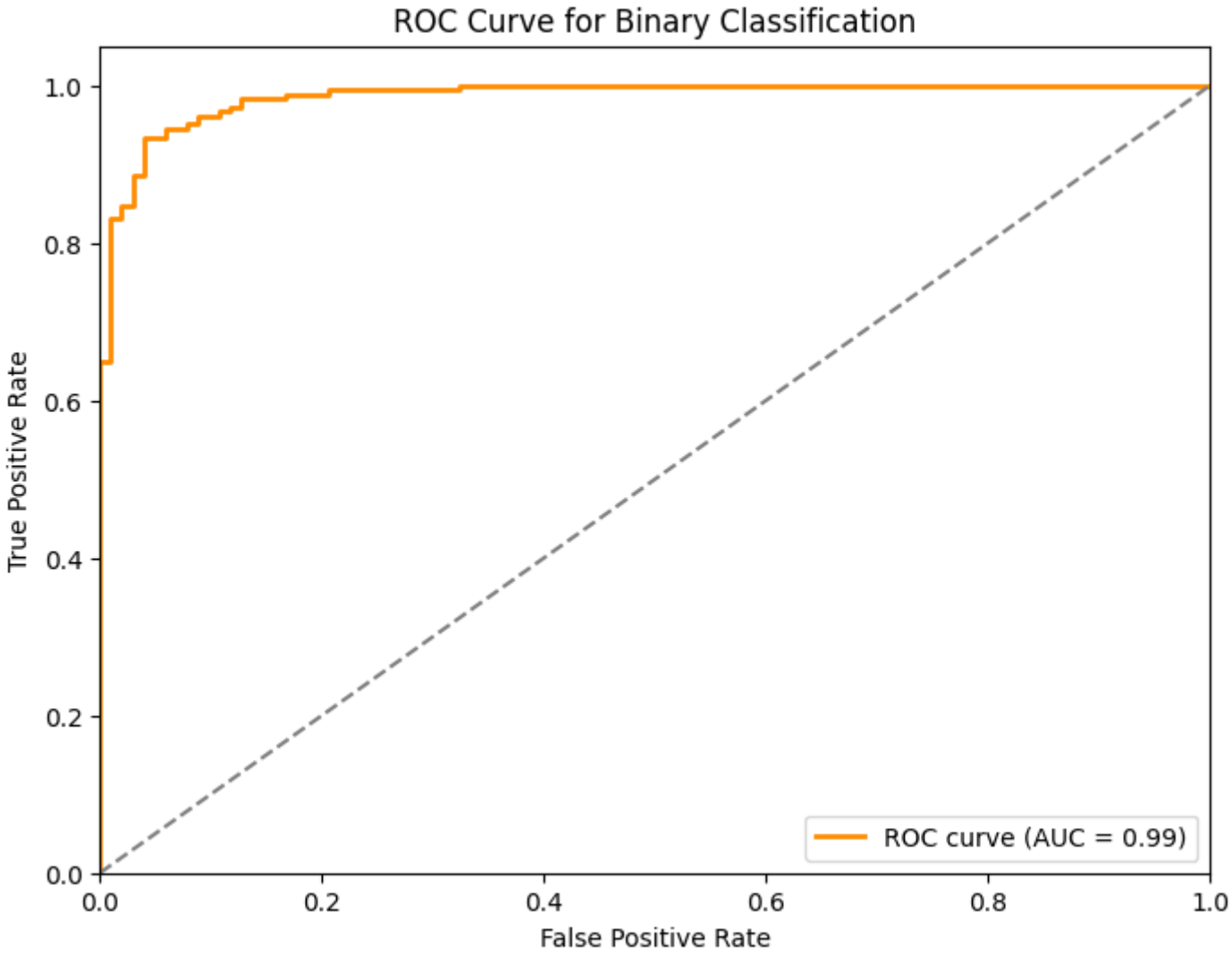
MLP Classifier

```
Got Maximum Accuracy for Test Size =  50 %
Maximum Accuracy =  95.43859649122807 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.90      0.98      0.94       102
           1       0.99      0.94      0.96       183

    accuracy                           0.95       285
   macro avg       0.94      0.96      0.95       285
weighted avg       0.96      0.95      0.95       285
```

Confusion Matrix:




MLP Classifier
Accuracy
95
94
93
92
91
30
35
40
45
50
55
60
65
70
Test Size



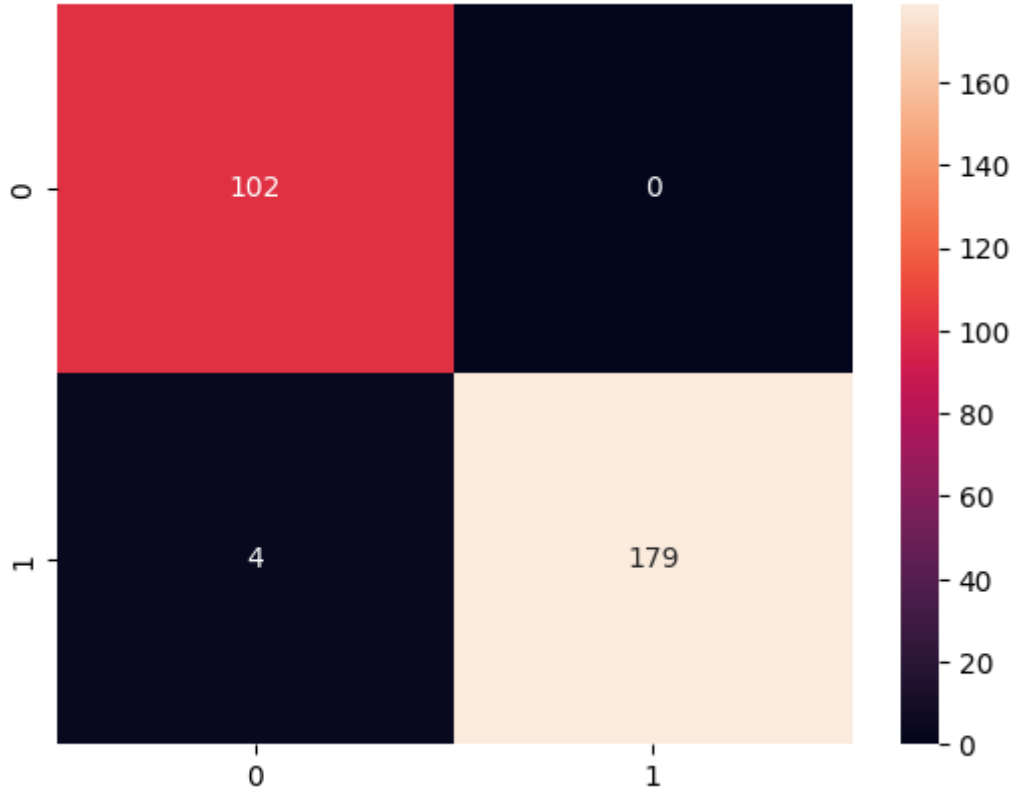
****

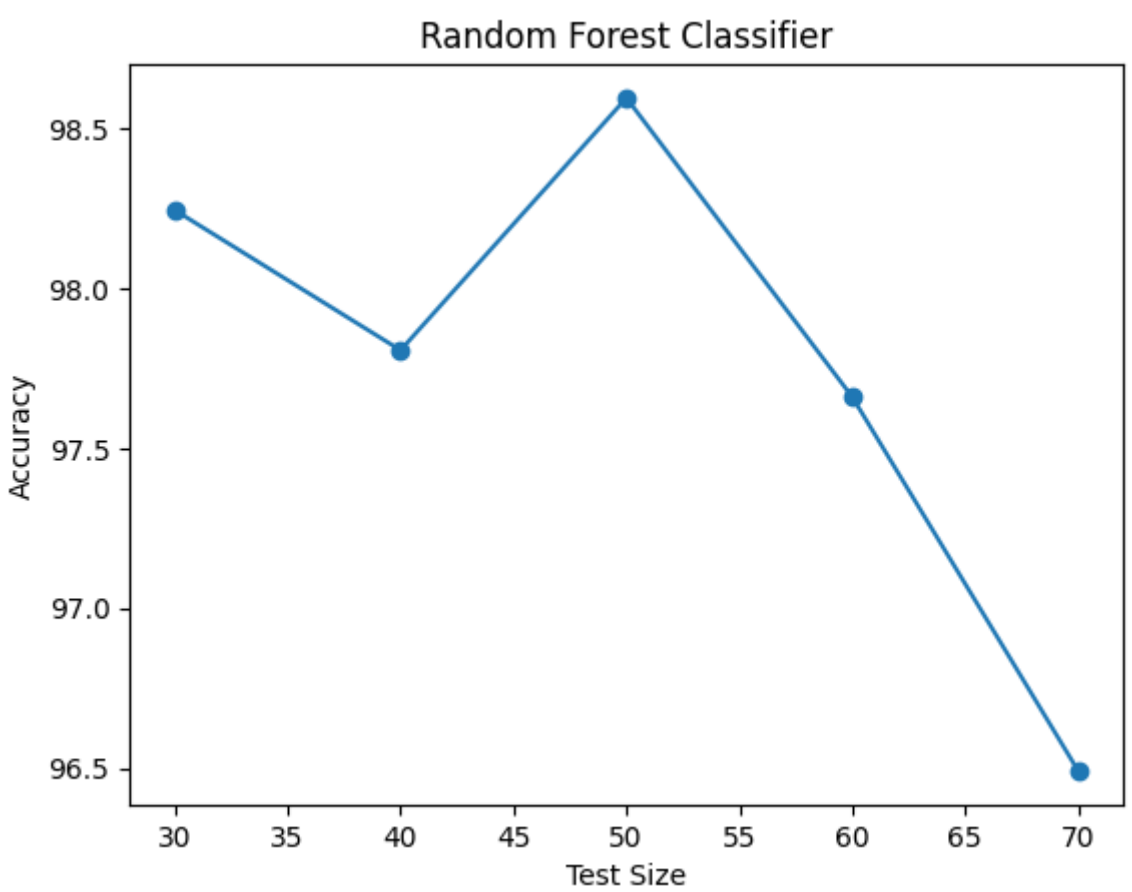
In [ ]:
```
_ = performOperation(X, y, model=RandomForestClassifier(), title="Random Forest Classifier", sampler=RandomOverSampler())
```

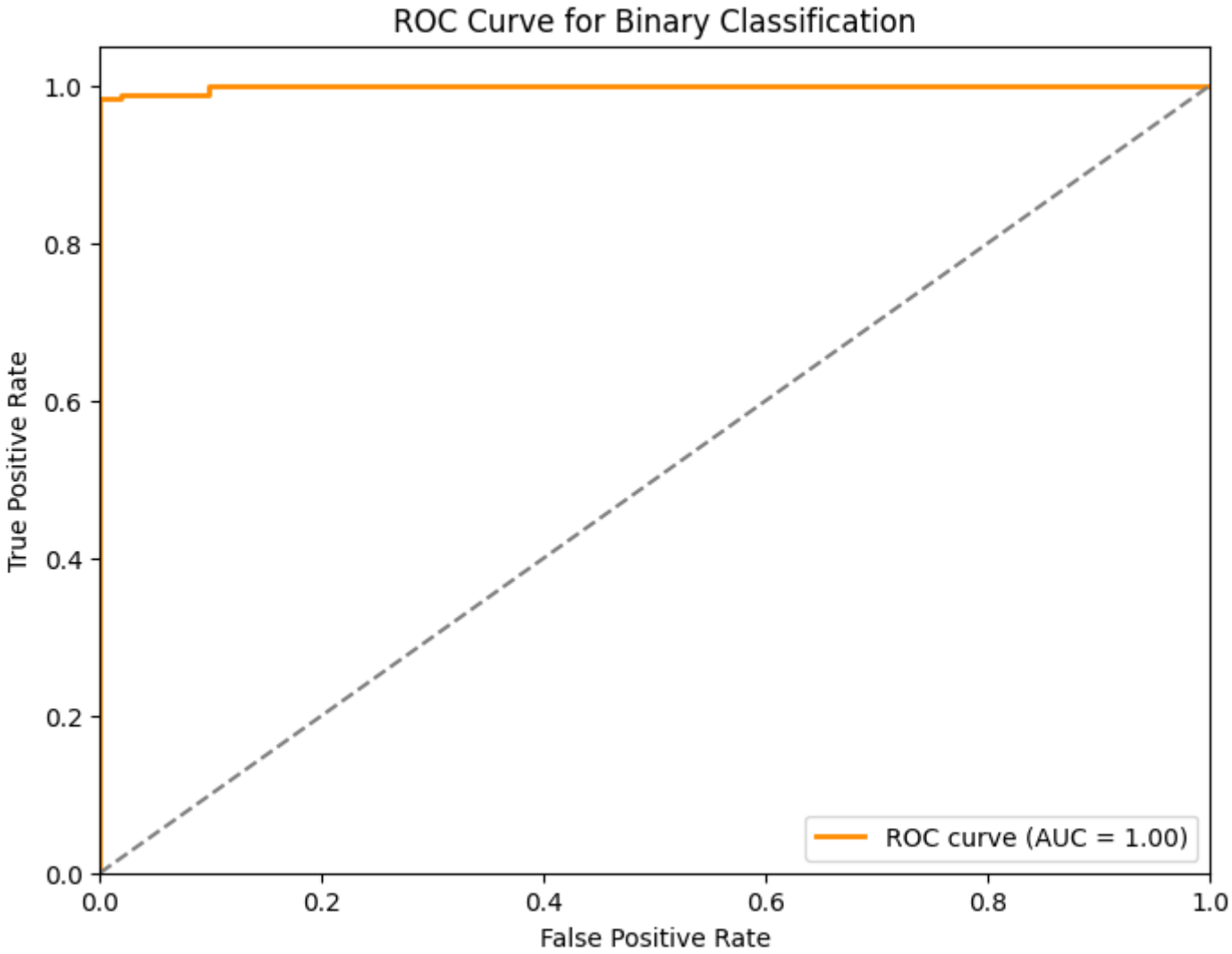
****

Random Forest Classifier

```
Got Maximum Accuracy for Test Size =  50 %
Maximum Accuracy =  98.59649122807016 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.96      1.00      0.98       102
           1       1.00      0.98      0.99       183

    accuracy                           0.99       285
   macro avg       0.98      0.99      0.98       285
weighted avg       0.99      0.99      0.99       285

Confusion Matrix:
```




Random Forest Classifier
Accuracy
98.5
98.0
97.5
97.0
96.5
30
35
40
45
50
55
60
65
70
Test Size

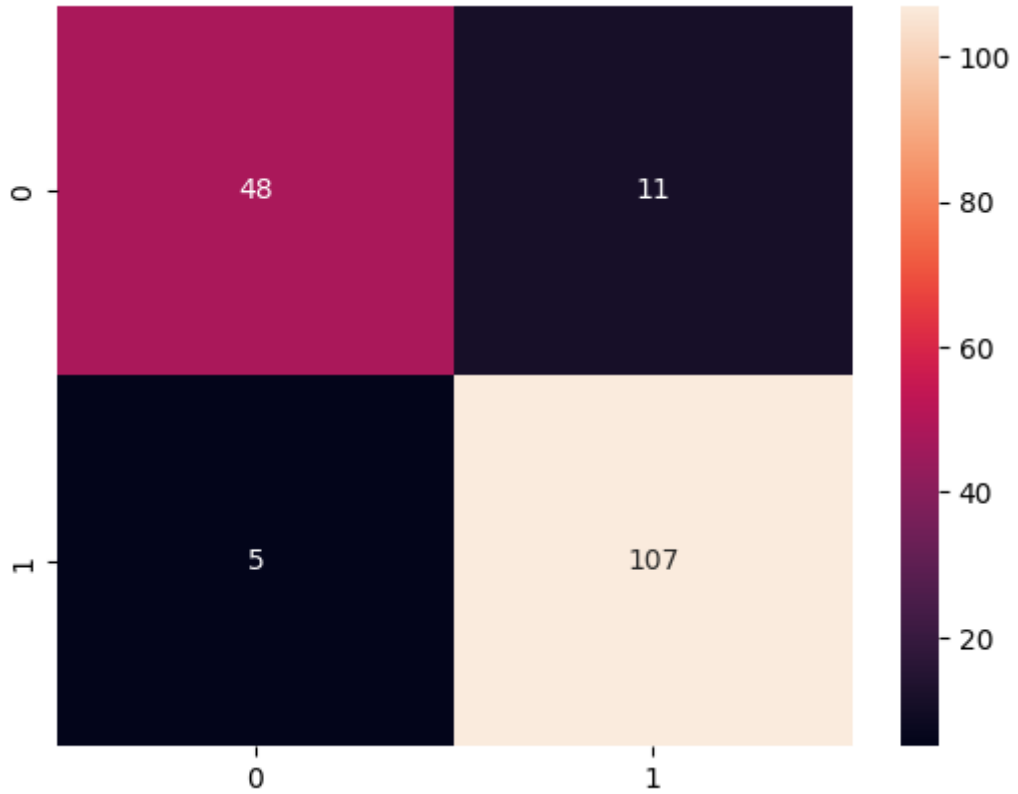


****

In [ ]:
```
pca = PCA(n_components=25)
X_pca = pca.fit_transform(X)
```

In [ ]:
```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier")
performOperationPCA(X_pca, y, model=SVC(kernel='sigmoid', gamma=0.01), test_split=0.3, sampler=RandomOverSampler(), scaler=StandardScaler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       0.91      0.81      0.86        59
           1       0.91      0.96      0.93       112

    accuracy                           0.91       171
   macro avg       0.91      0.88      0.89       171
weighted avg       0.91      0.91      0.91       171

Confusion Matrix:
```



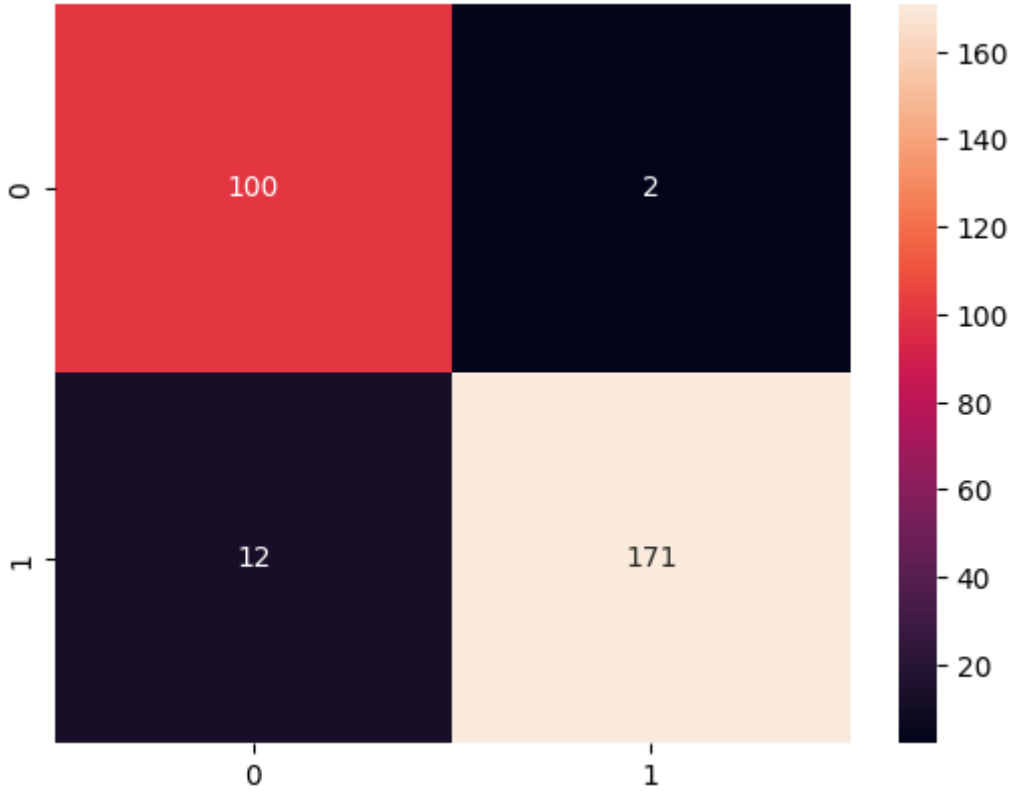
In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier")
performOperationPCA(X_pca, y, model=MLPClassifier(max_iter=1000, learning_rate='adaptive'), test_split=0.5, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       0.89      0.98      0.93       102
```

```
           1       0.99      0.93      0.96       183

    accuracy                           0.95       285
   macro avg       0.94      0.96      0.95       285
weighted avg       0.95      0.95      0.95       285
```

Confusion Matrix:

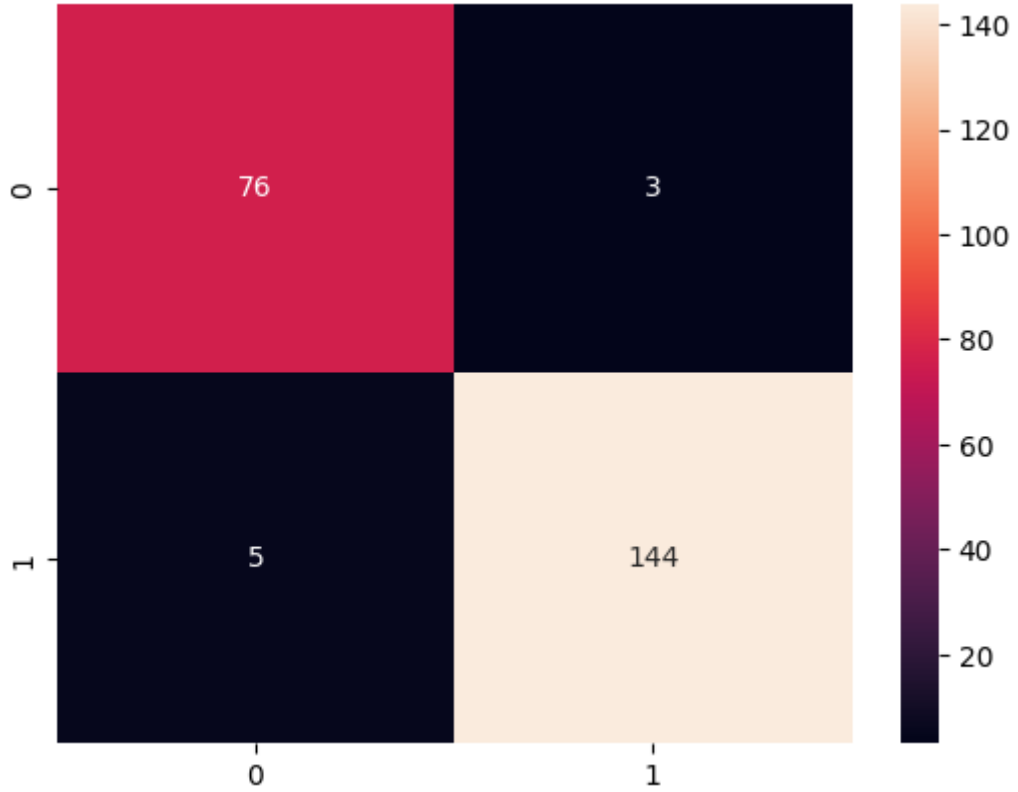


In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier")
performOperationPCA(X_pca, y, model=RandomForestClassifier(), test_split=0.4, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95        79
           1       0.98      0.97      0.97       149

    accuracy                           0.96       228
   macro avg       0.96      0.96      0.96       228
weighted avg       0.97      0.96      0.97       228
```

```
Confusion Matrix:
```



The Random Forest classifier achieves a peak accuracy of approximately 99% at a test size of 40%.

However, the accuracy drops to around 96% after the application of PCA.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
```

In [ ]:
```
!gdown 16eCflW27A3ZBaCBOh3bthU3MHaEhooES
```

```
Downloading...
From: https://drive.google.com/uc?id=16eCflW27A3ZBaCBOh3bthU3MHaEhooES
To: /content/ionosphere.data
100% 76.5k/76.5k [00:00<00:00, 74.6MB/s]
```

In [ ]:
```
cols = [f'Attribute{i}' for i in range(1, 35)]
cols.append('class')
df = pd.read_csv("ionosphere.data", names=cols)
```

In [ ]:
```
df.drop_duplicates(subset=None, keep='first', inplace=True)
```

In [ ]:
```
X = df.drop('class', axis=1)
y = df['class']
y = np.where(y == 'b', 0, 1)
```

In [ ]:
```
# Function for splitting, scaling, and sampling the dataset
def preprocessing(X, y, scaler, sampler, test_split):
  if(scaler != None):
    X = scaler.fit_transform(X)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_split, random_state=10)
  if(sampler != None):
    X_train, y_train = RandomOverSampler().fit_resample(X_train, y_train)
  return X_train, y_train, X_test, y_test
```

In [ ]:

```
# Function for training and prediction
def trainAndPredict(model, X_train, y_train, X_test, y_test):
  model = model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  report = classification_report(y_test, y_pred)
  conf_matrix = confusion_matrix(y_test, y_pred)
  return accuracy, report, conf_matrix
```

In [ ]:

```
# Function for generating an image illustrating Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC)
def generateROCAndAUC(X, y, test_size, model, scaler, sampler):
  X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=scaler, sampler=sampler, test_split=test_size)
  model.fit(X_train, y_train)
  if hasattr(model, "predict_proba"):
      probas = model.predict_proba(X_test)
      prob_positive_class = probas[:, 1]
  else:
      decision_function = model.decision_function(X_test)
      prob_positive_class = (decision_function - decision_function.min()) / (decision_function.max() - decision_function.min())

  fpr, tpr, _ = roc_curve(y_test, prob_positive_class)
  roc_auc = auc(fpr, tpr)

  plt.figure(figsize=(8, 6))
  plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (AUC = {roc_auc:.2f})')
  plt.plot([0, 1], [0, 1], color='gray', 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(f'ROC Curve for Binary Classification')
  plt.legend(loc='lower right')
  plt.show()
```

In [ ]:

```
def performOperation(X, y, model, title, scaler=None, sampler=None):
  print("\n\n**********************************************************************************************************************
  print(title);
  graph_x = []
  graph_y = []
  max_accuracy = 0
  test_size = 0
  final_report = None
  final_conf_matrix = None
  for test_split in range(3, 8):
```

```
    X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=scaler, sampler=sampler,  test_split=test_split*0.1)
    accuracy, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
    if max_accuracy < accuracy:
      test_size = test_split*0.1
      max_accuracy = accuracy
      final_report = report
      final_conf_matrix = conf_matrix
    graph_x.append(test_split*10)
    graph_y.append(accuracy*100)
  print("Got Maximum Accuracy for Test Size = ", int(test_size*100), "%")
  print("Maximum Accuracy = ", max_accuracy*100, "%")
  print("Performance Evolution:")
  print(final_report)
  print("Confusion Matrix:")
  sns.heatmap(final_conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
  plt.plot(graph_x, graph_y, marker='o')
  plt.xlabel('Test Size')
  plt.ylabel('Accuracy')
  plt.title(title)
  plt.show()
  print("\n")
  generateROCAndAUC(X, y, test_size=test_size, model=model, scaler=scaler, sampler=sampler)
  print("\n\n**********************************************************************************************************************************")
  return graph_y
```

In [ ]:

```
def performOperationPCA(X, y, model, test_split, sampler=None, scaler=None):
  X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=None, sampler=sampler,  test_split=test_split)
  _, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
  print("Performance Evolution:")
  print(report)
  print("Confusion Matrix:")
  sns.heatmap(conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
```

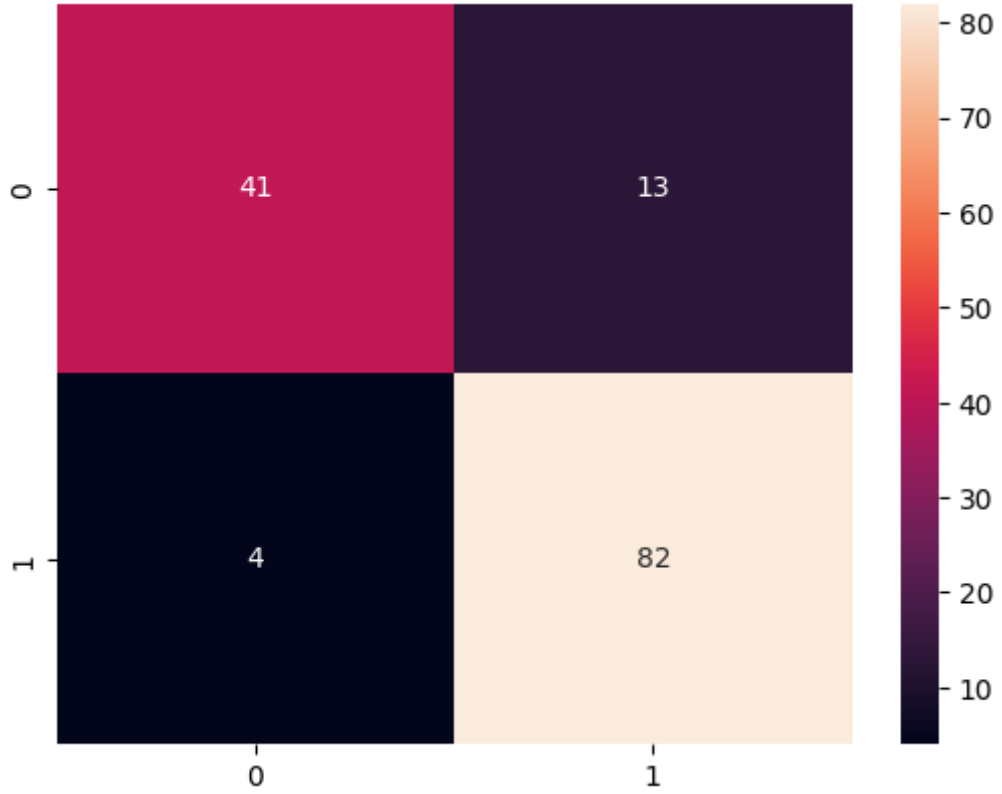
In [ ]:

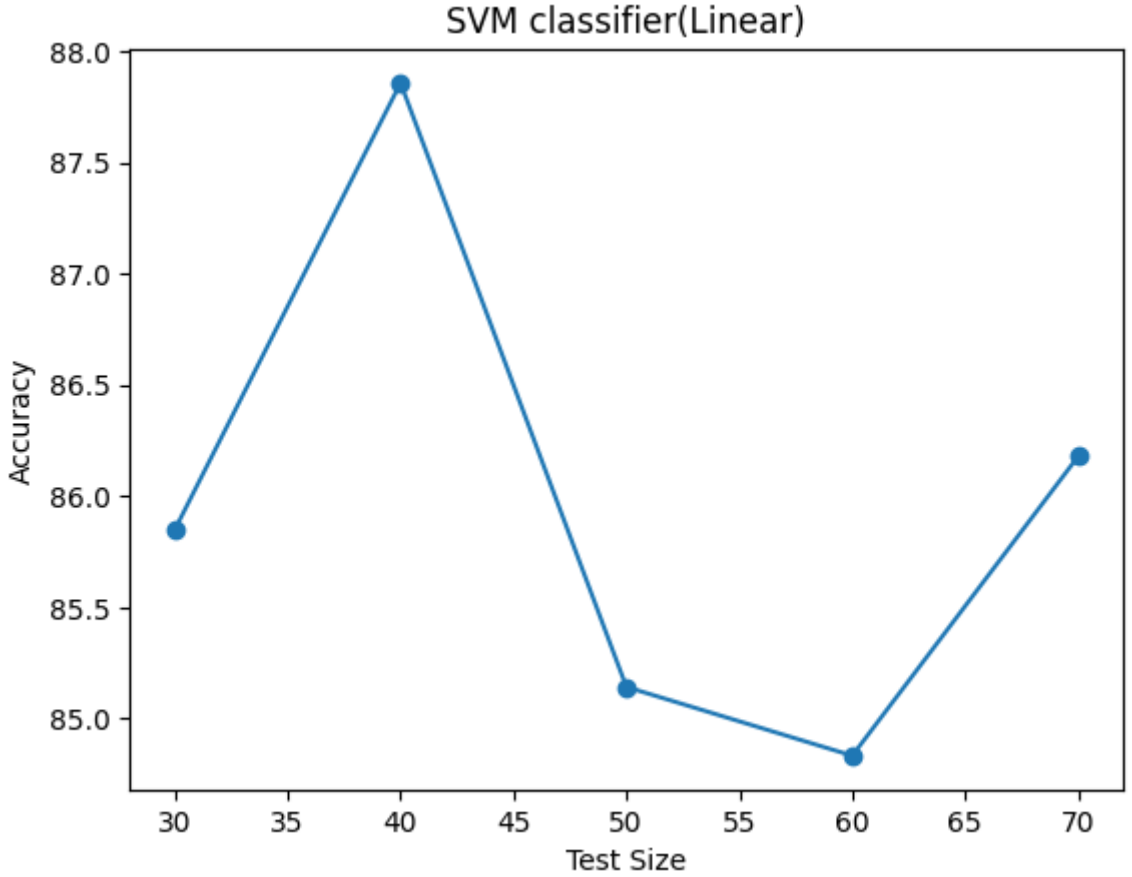
```
svm_data = []
svm_data.append(performOperation(X, y, model=SVC(kernel='linear'), title="SVM classifier(Linear)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='poly'), title="SVM classifier(Polynomial)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='sigmoid', gamma=0.01), title="SVM classifier(Sigmoid)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='rbf'), title="SVM classifier(Gaussian)", sampler=RandomOverSampler()))
```

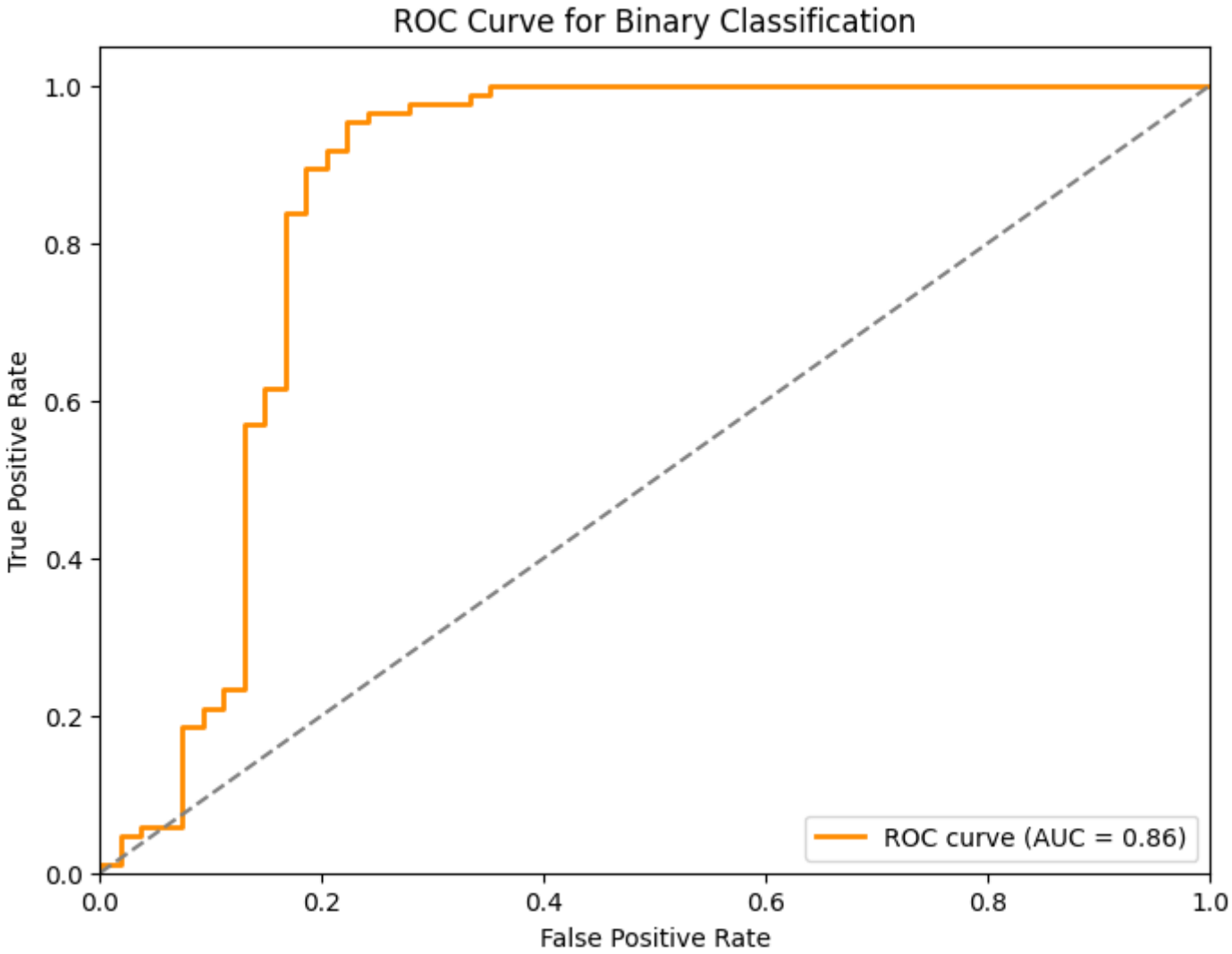
```
**********************************************************************************************************************************
**********************************************************
```

```
SVM classifier(Linear)
Got Maximum Accuracy for Test Size =  40 %
Maximum Accuracy =  87.85714285714286 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.91      0.76      0.83        54
           1       0.86      0.95      0.91        86

    accuracy                           0.88       140
   macro avg       0.89      0.86      0.87       140
weighted avg       0.88      0.88      0.88       140
```

Confusion Matrix:




SVM classifier(Linear)
88.0
87.5
87.0
86.5
86.0
85.5
85.0
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

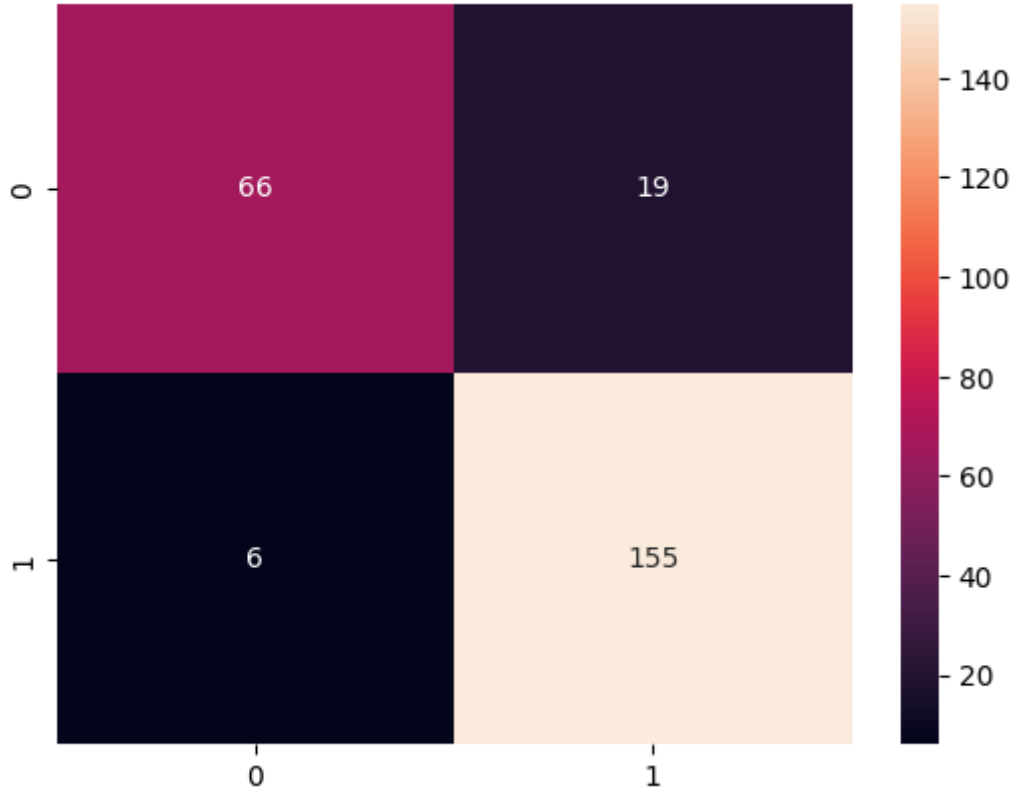
****

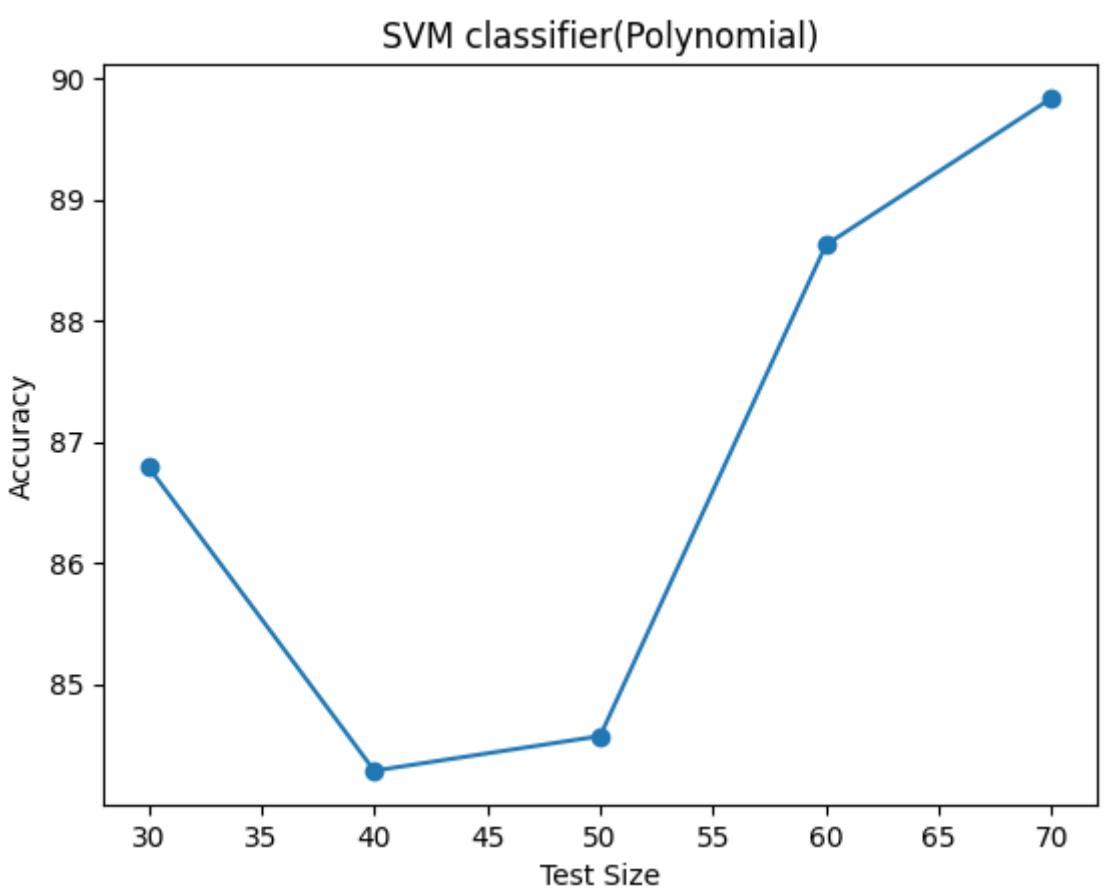
SVM classifier(Polynomial)
Got Maximum Accuracy for Test Size = 70 %
Maximum Accuracy = 89.83739837398373 %

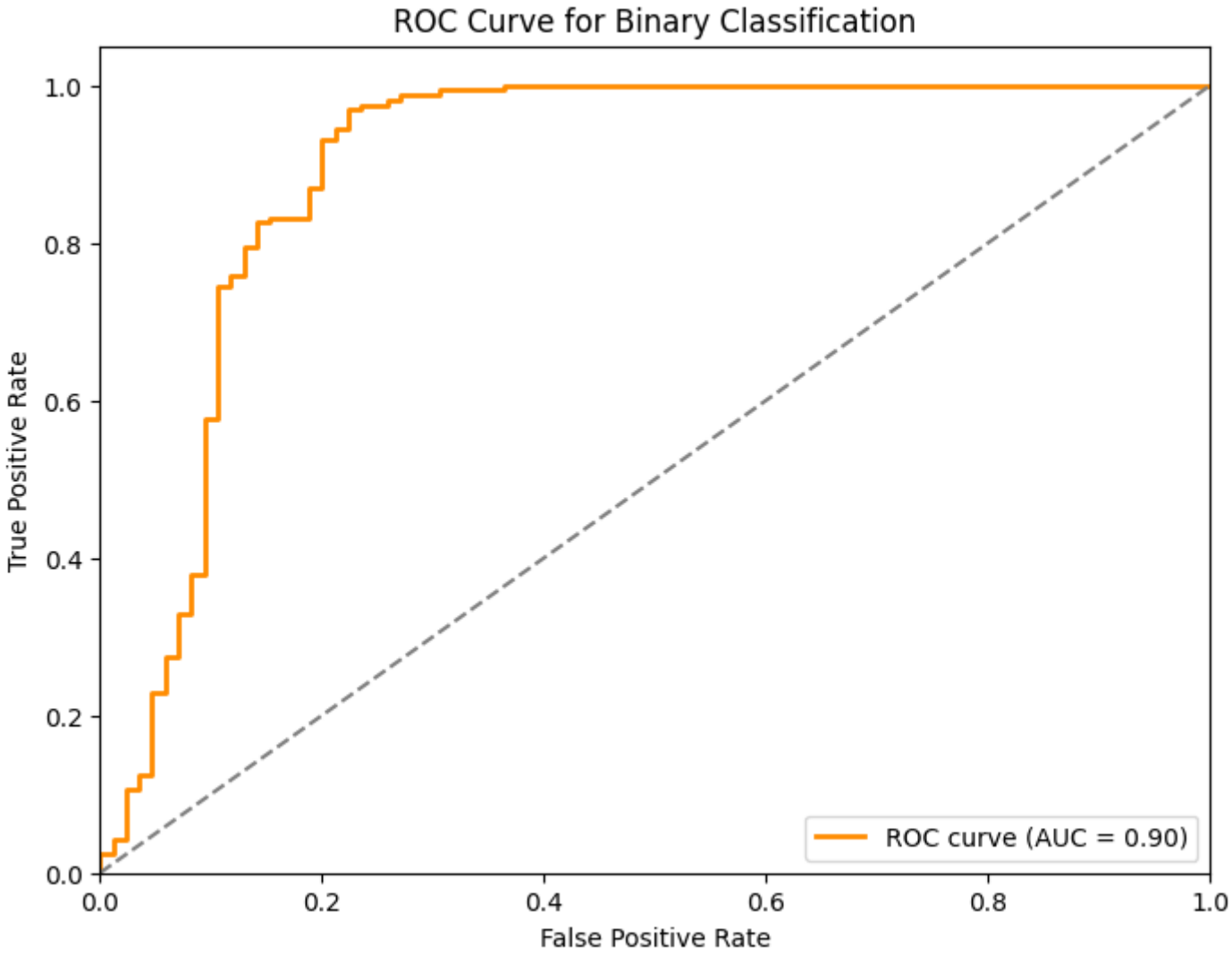
Performance Evolution:

	precision	recall	f1-score	support
0	0.92	0.78	0.84	85
1	0.89	0.96	0.93	161
accuracy			0.90	246
macro avg	0.90	0.87	0.88	246
weighted avg	0.90	0.90	0.90	246

Confusion Matrix:




SVM classifier(Polynomial)
Accuracy
Test Size
90
89
88
87
86
85
30
35
40
45
50
55
60
65
70



****
****

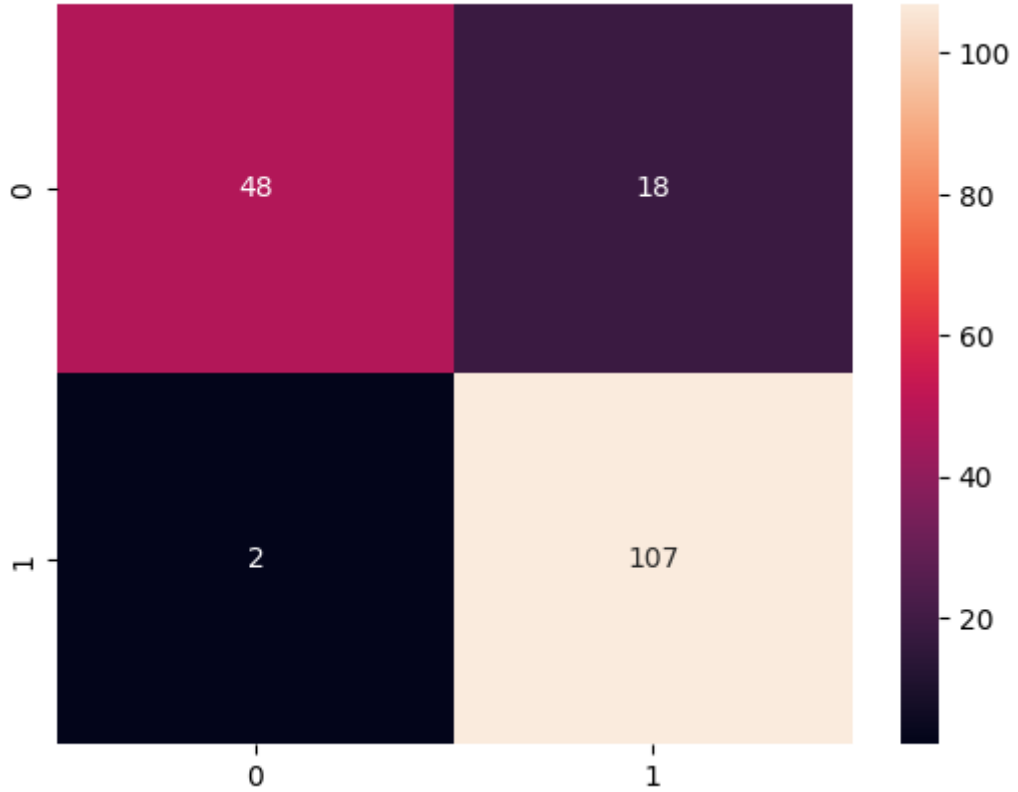
****
****

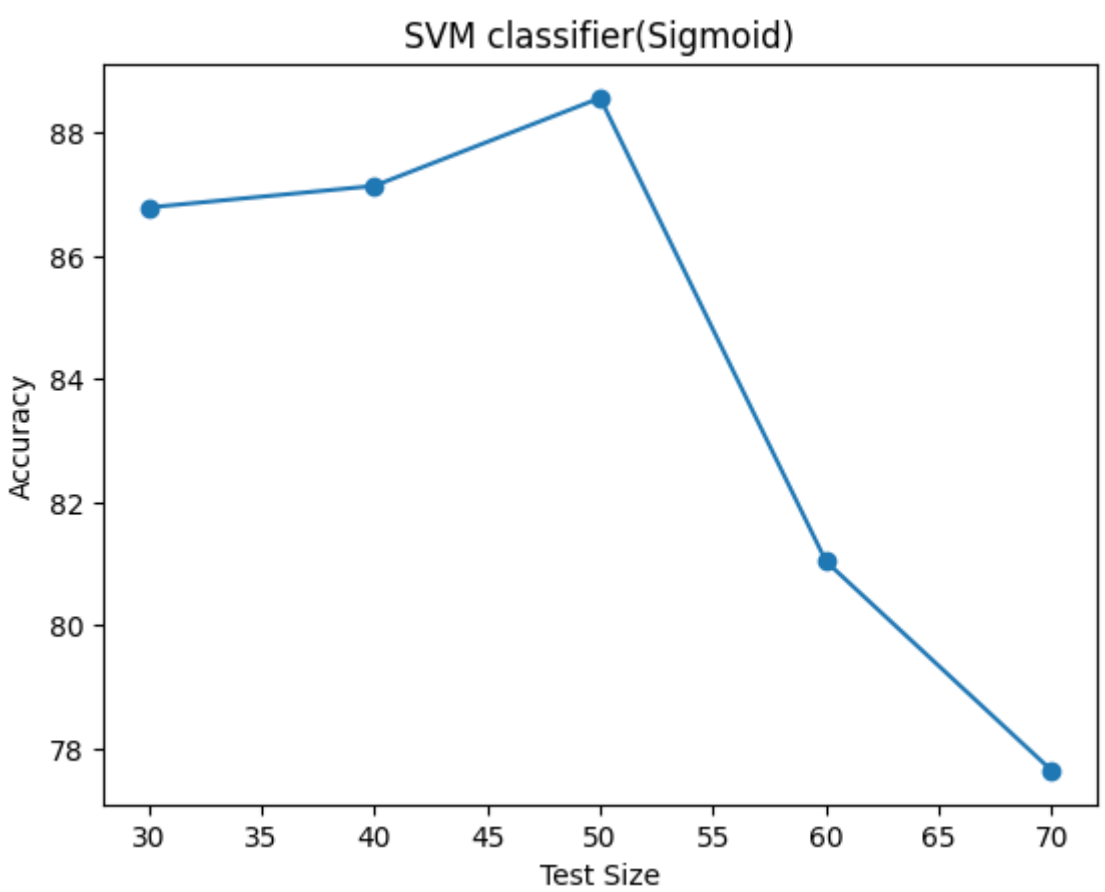
SVM classifier(Sigmoid)
Got Maximum Accuracy for Test Size = 50 %
Maximum Accuracy = 88.57142857142857 %

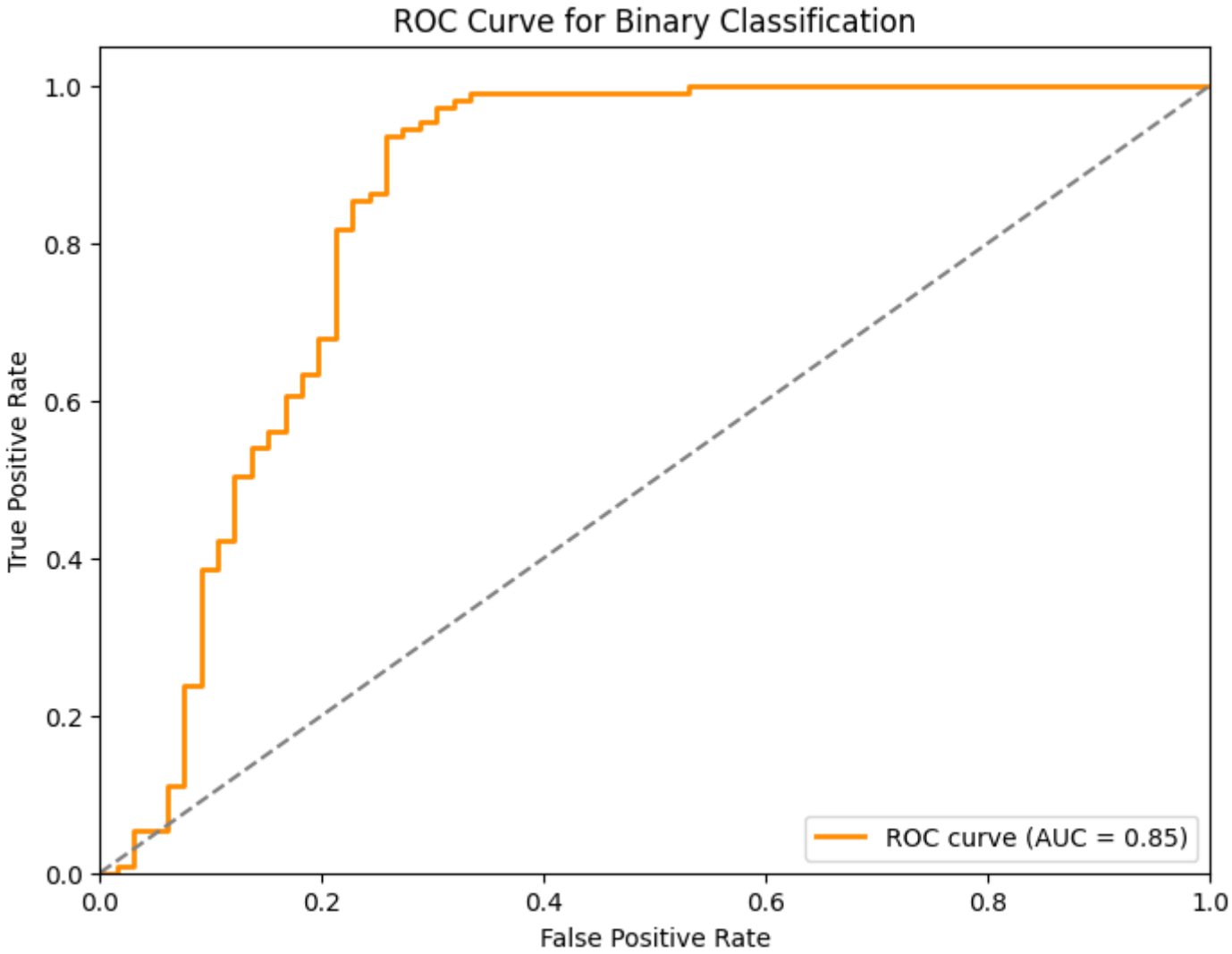
Performance Evolution:

	precision	recall	f1-score	support
0	0.96	0.73	0.83	66
1	0.86	0.98	0.91	109
accuracy			0.89	175
macro avg	0.91	0.85	0.87	175
weighted avg	0.90	0.89	0.88	175

Confusion Matrix:




SVM classifier(Sigmoid)
Accuracy
88
86
84
82
80
78
30
35
40
45
50
55
60
65
70
Test Size



****

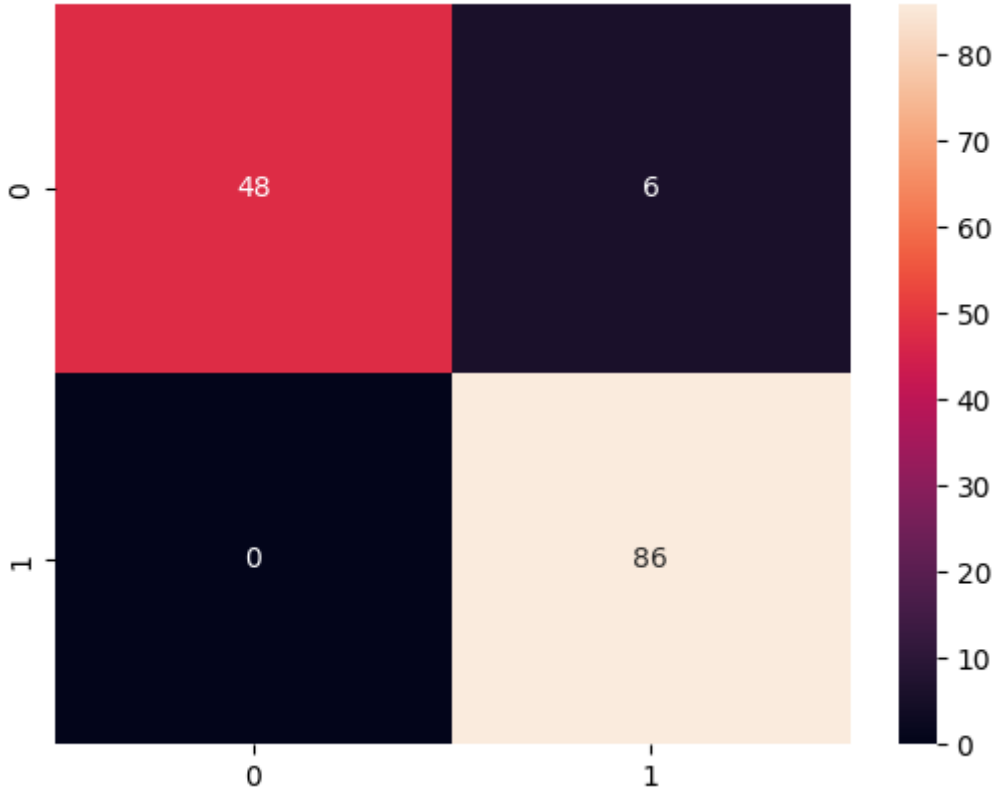
****

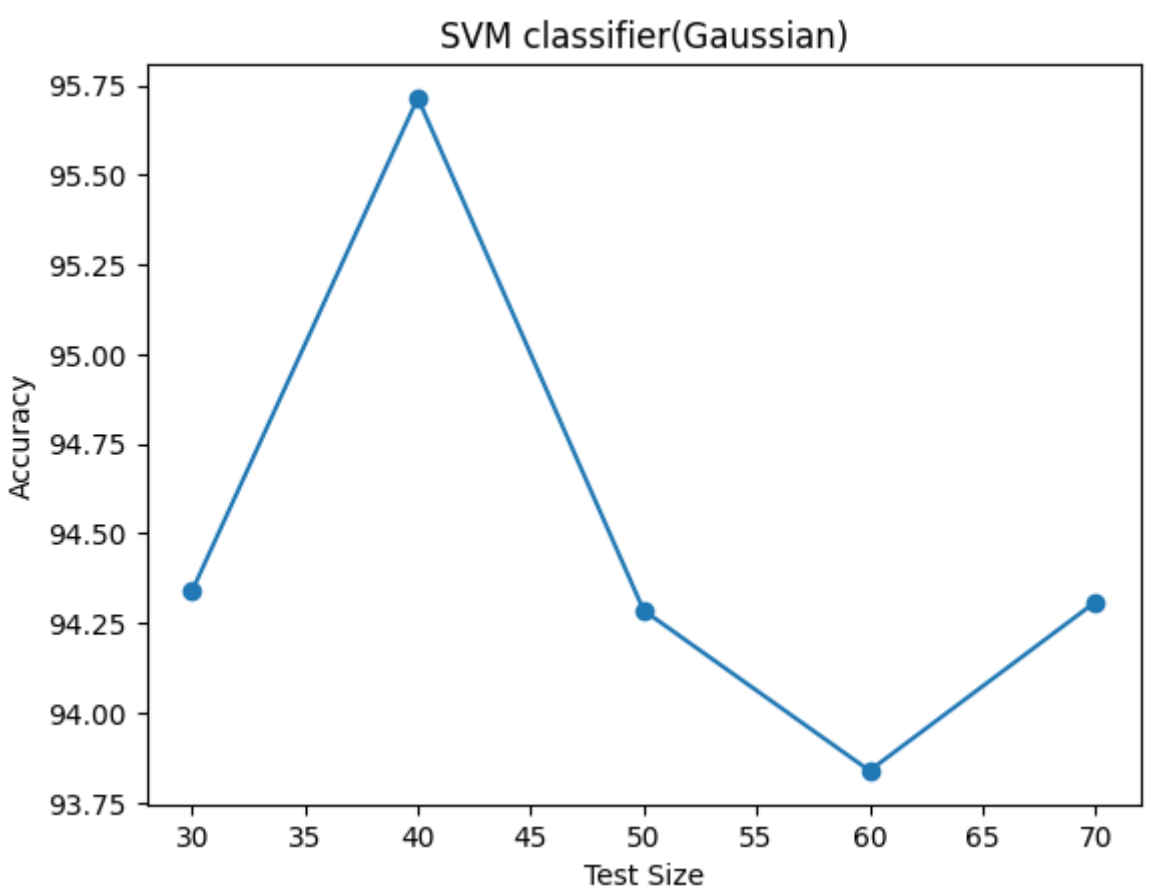
SVM classifier(Gaussian)
Got Maximum Accuracy for Test Size = 40 %
Maximum Accuracy = 95.71428571428572 %

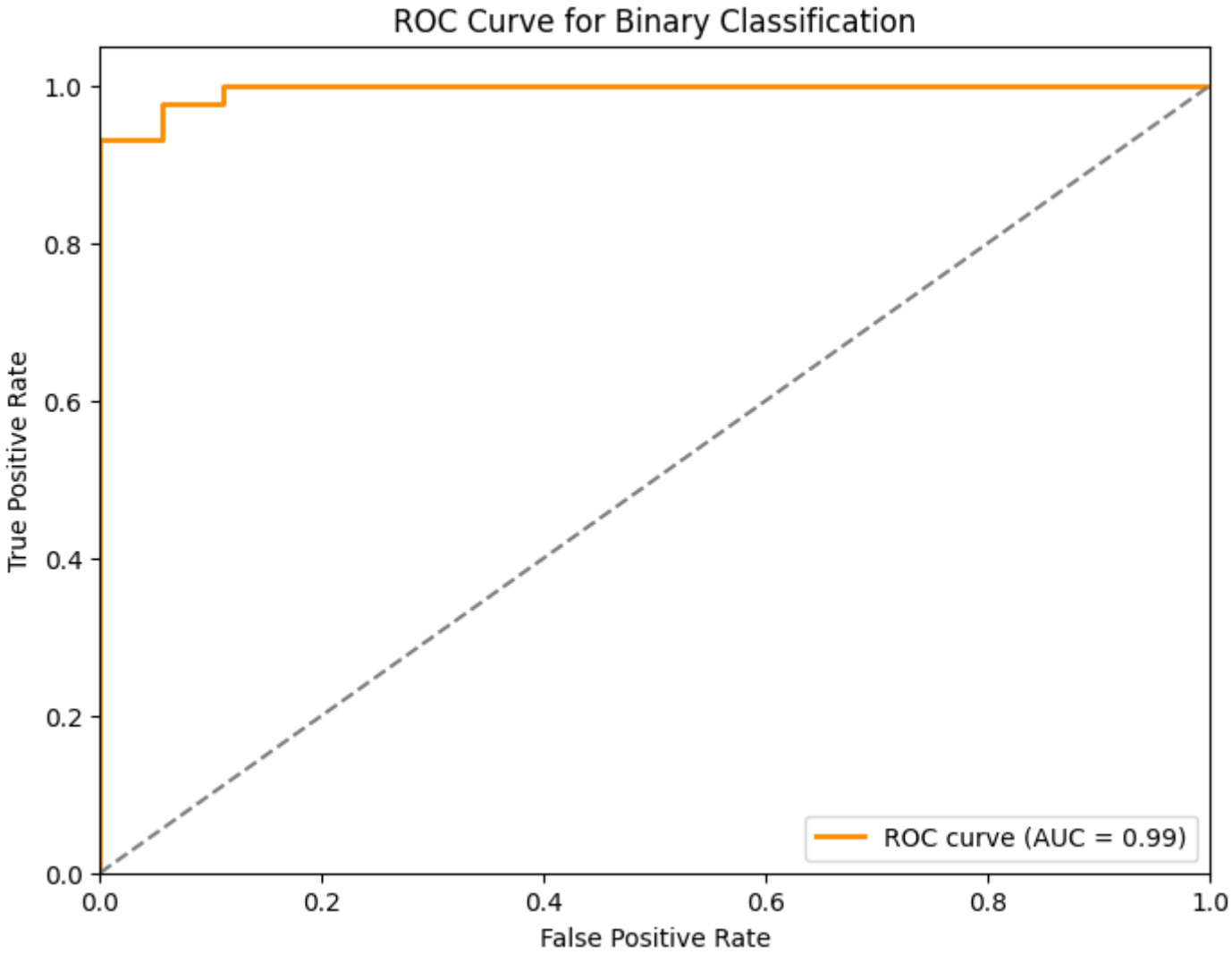
Performance Evolution:

	precision	recall	f1-score	support
0	1.00	0.89	0.94	54
1	0.93	1.00	0.97	86
accuracy			0.96	140
macro avg	0.97	0.94	0.95	140
weighted avg	0.96	0.96	0.96	140

Confusion Matrix:




SVM classifier(Gaussian)
95.75
95.50
95.25
95.00
94.75
94.50
94.25
94.00
93.75
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

In [ ]:
```
table = pd.DataFrame(svm_data, index=["SVM (Linear)", 'SVM (Polynomial)', 'SVM (Sigmoid)', 'SVM (Gaussian)'], columns = [f"Test Size = {10 *
print(table)
```

```
                 Test Size = 30%  Test Size = 40%  Test Size = 50%  \
SVM (Linear)           85.849057        87.857143        85.142857
SVM (Polynomial)       86.792453        84.285714        84.571429
SVM (Sigmoid)          86.792453        87.142857        88.571429
SVM (Gaussian)         94.339623        95.714286        94.285714
```

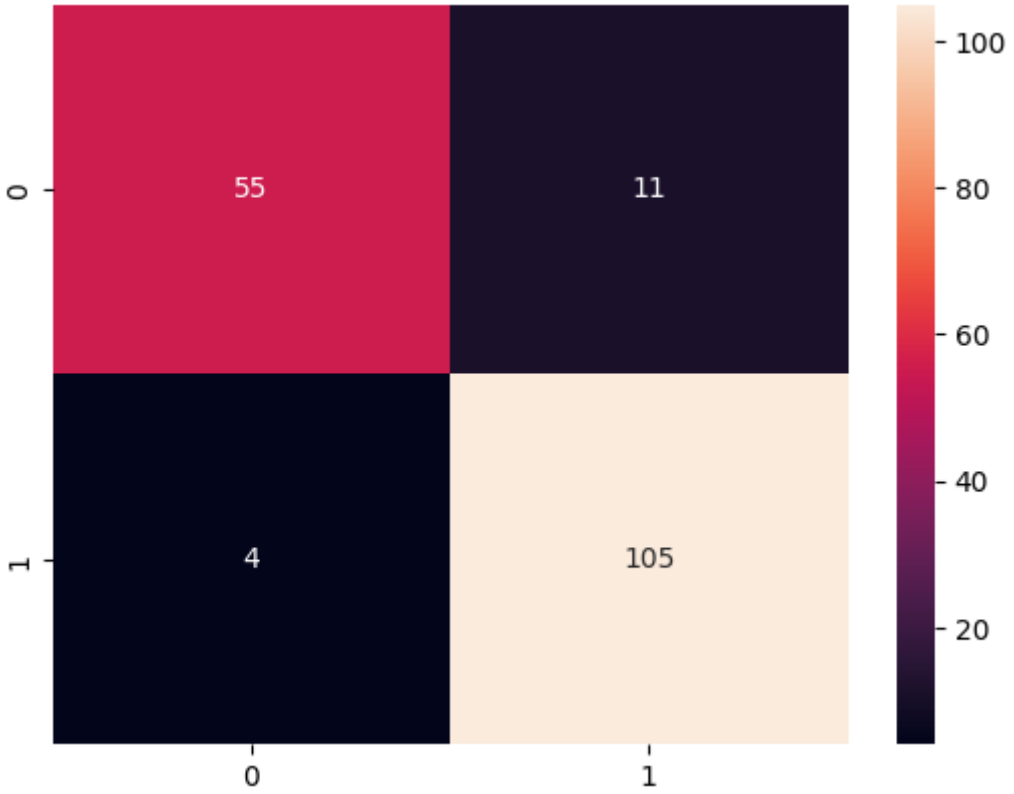
```
                  Test Size = 60%  Test Size = 70%
SVM (Linear)            84.834123        86.178862
SVM (Polynomial)        88.625592        89.837398
SVM (Sigmoid)           81.042654        77.642276
SVM (Gaussian)          93.838863        94.308943
```

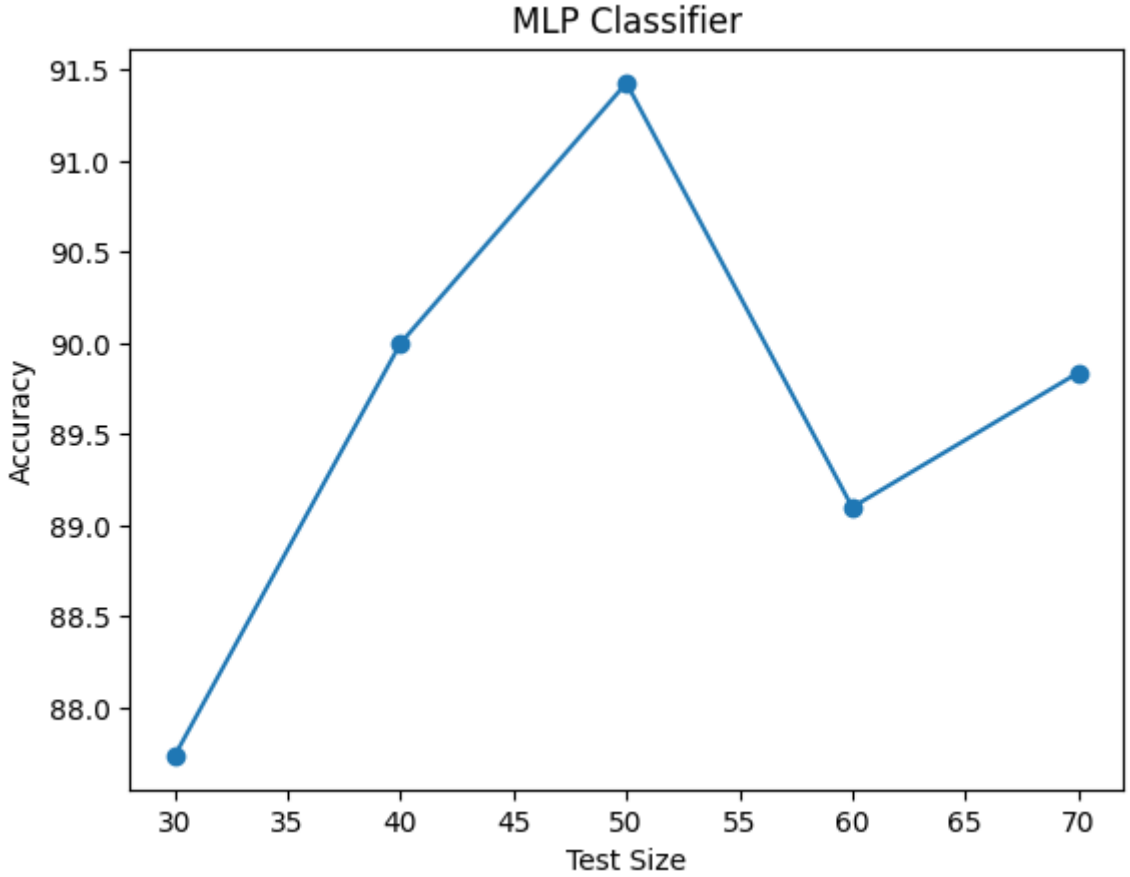
In [ ]:

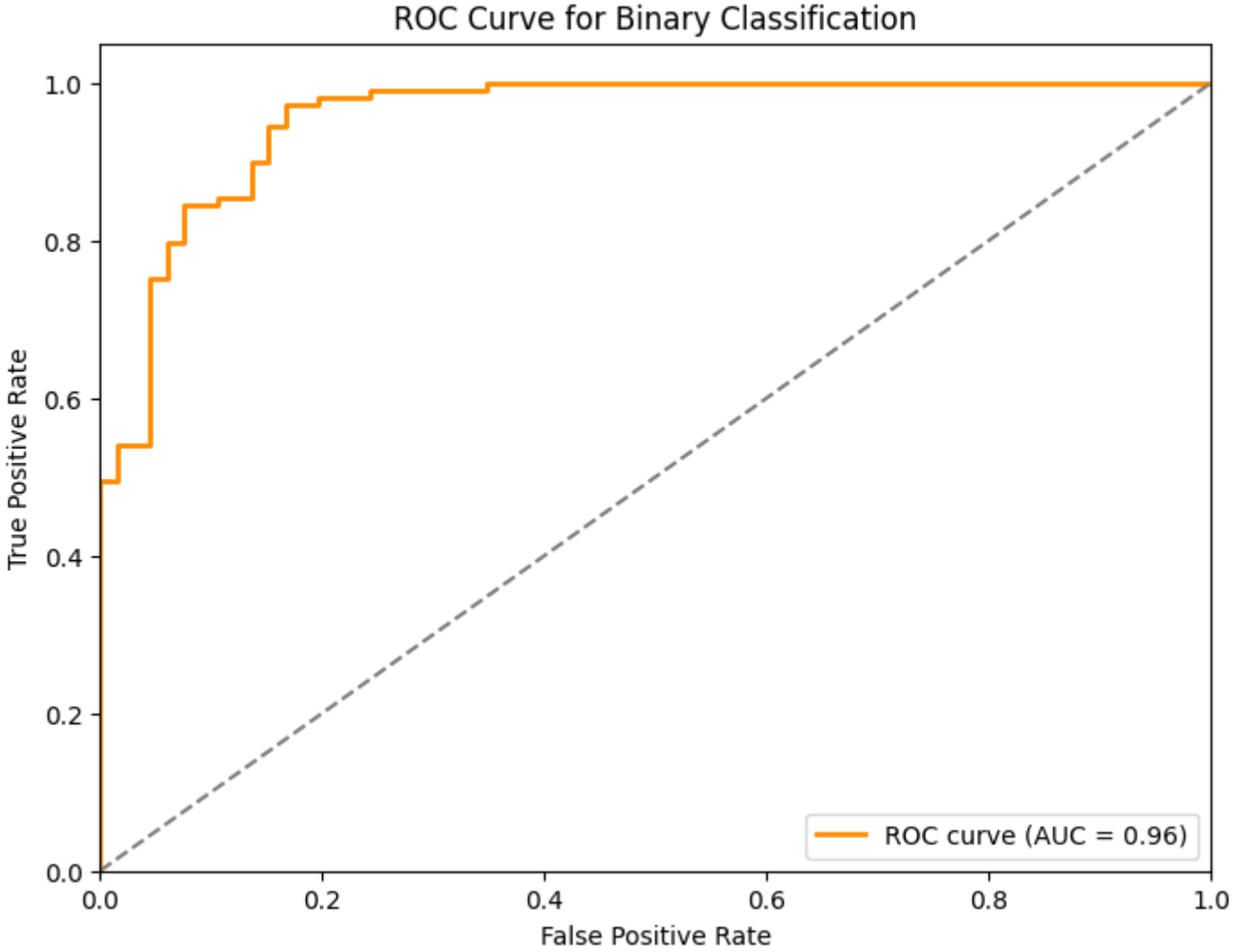
```
_ = performOperation(X, y, model=MLPClassifier(max_iter=1000, learning_rate='adaptive'), title="MLP Classifier", sampler=RandomOverSampler())
```

```
**************************************************************************************************************************************************
**************************************************
```

```
MLP Classifier
Got Maximum Accuracy for Test Size =  50 %
Maximum Accuracy =  91.42857142857143 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.93      0.83      0.88        66
           1       0.91      0.96      0.93       109

    accuracy                           0.91       175
   macro avg       0.92      0.90      0.91       175
weighted avg       0.92      0.91      0.91       175

Confusion Matrix:
```


55
11
4
105
0
1
0
1
100
80
60
40
20


MLP Classifier
Accuracy
Test Size
91.5
91.0
90.5
90.0
89.5
89.0
88.5
88.0
30
35
40
45
50
55
60
65
70



****

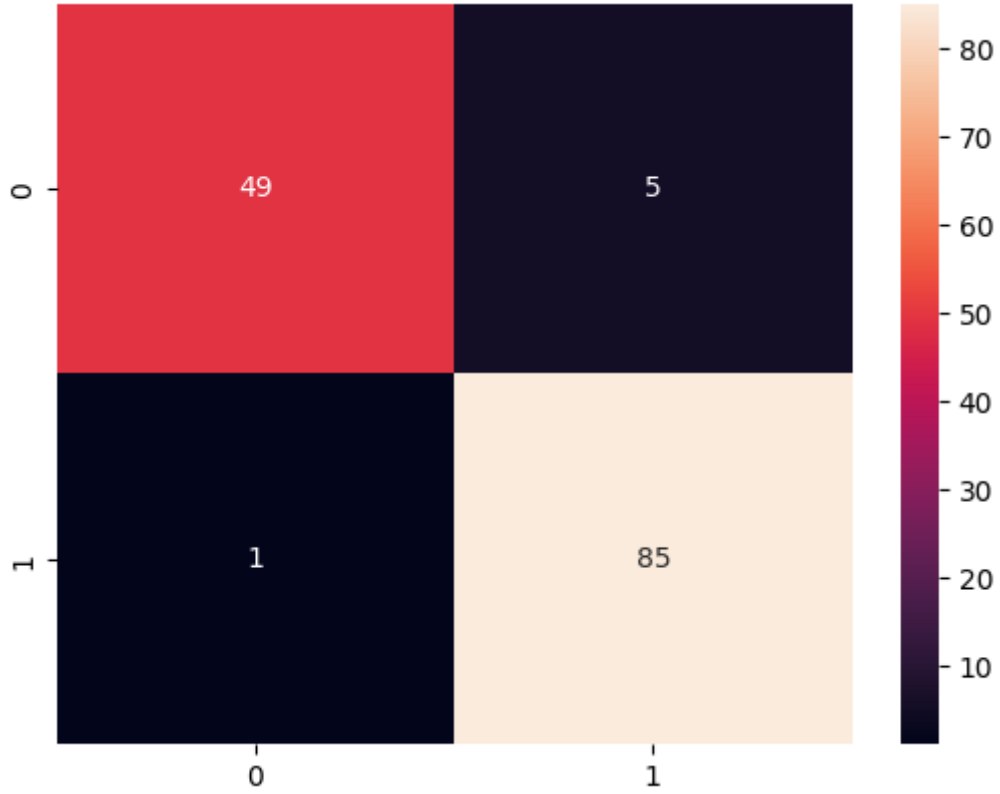
In [ ]:
```
_ = performOperation(X, y, model=RandomForestClassifier(), title="Random Forest Classifier", sampler=RandomOverSampler())
```

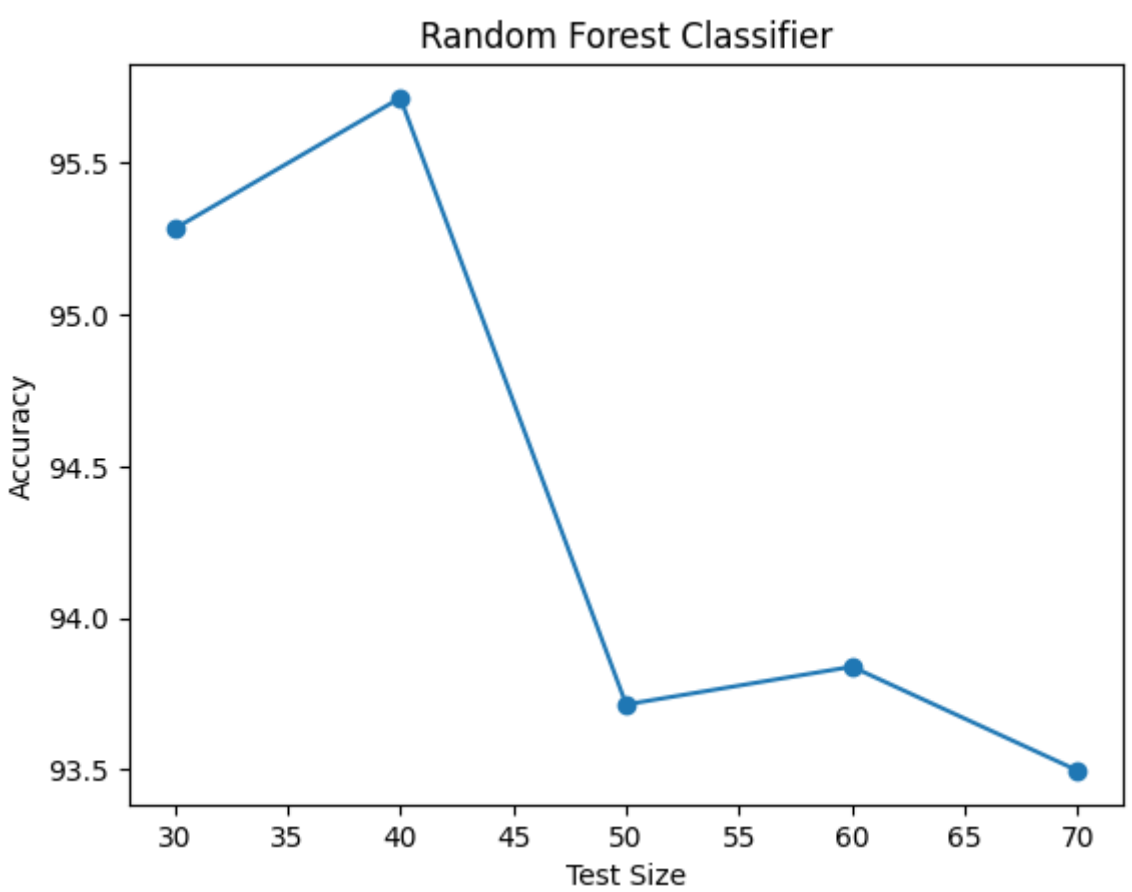
****

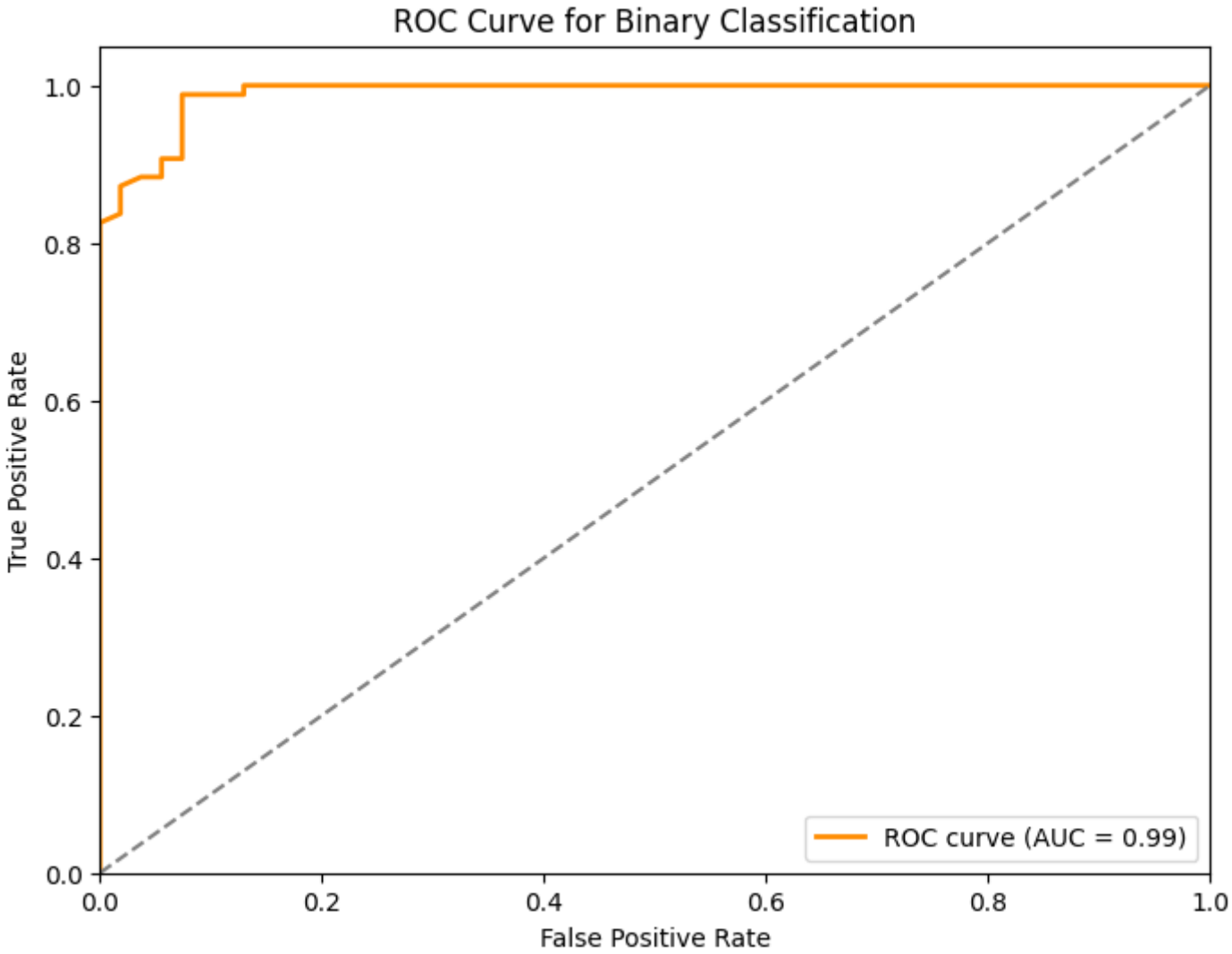
Random Forest Classifier

```
Got Maximum Accuracy for Test Size =  40 %
Maximum Accuracy =  95.71428571428572 %
Performance Evolution:
              precision    recall  f1-score   support

           0       0.98      0.91      0.94        54
           1       0.94      0.99      0.97        86

    accuracy                           0.96       140
   macro avg       0.96      0.95      0.95       140
weighted avg       0.96      0.96      0.96       140
```

Confusion Matrix:




Random Forest Classifier
95.5
95.0
94.5
94.0
93.5
Accuracy
30
35
40
45
50
55
60
65
70
Test Size

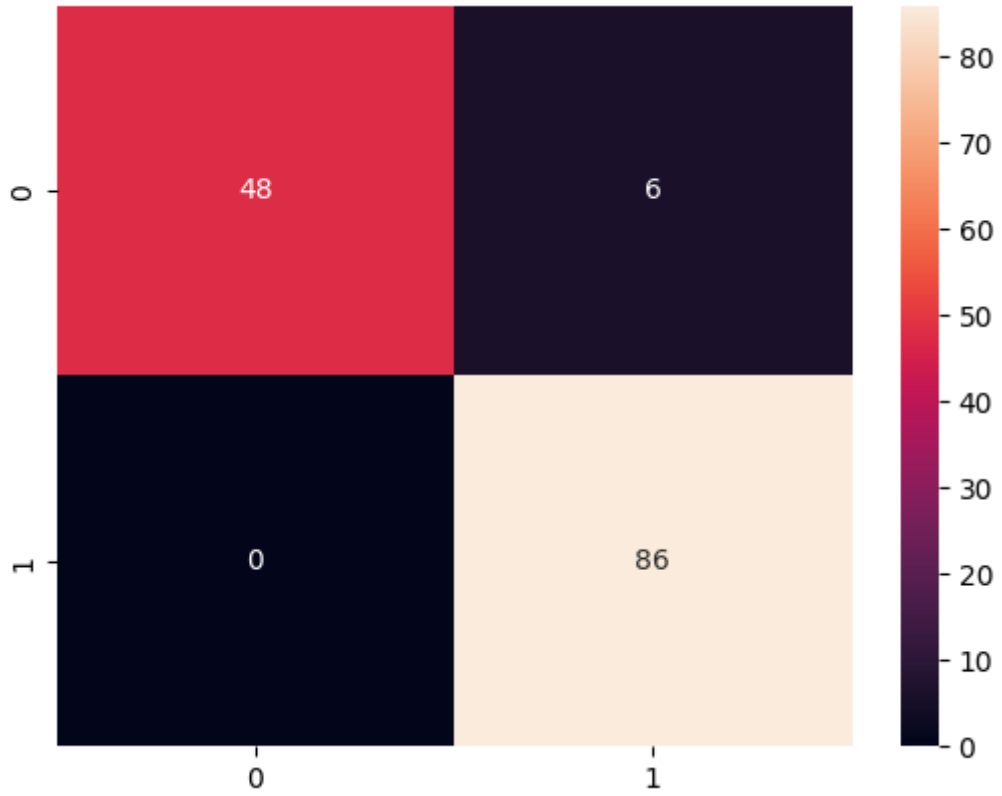


****

In [ ]:
```
pca = PCA(n_components=30)
X_pca = pca.fit_transform(X)
```

In [ ]:
```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier")
performOperationPCA(X_pca, y, model=SVC(kernel='rbf'), test_split=0.4, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       1.00      0.89      0.94        54
           1       0.93      1.00      0.97        86

    accuracy                           0.96       140
   macro avg       0.97      0.94      0.95       140
weighted avg       0.96      0.96      0.96       140

Confusion Matrix:
```



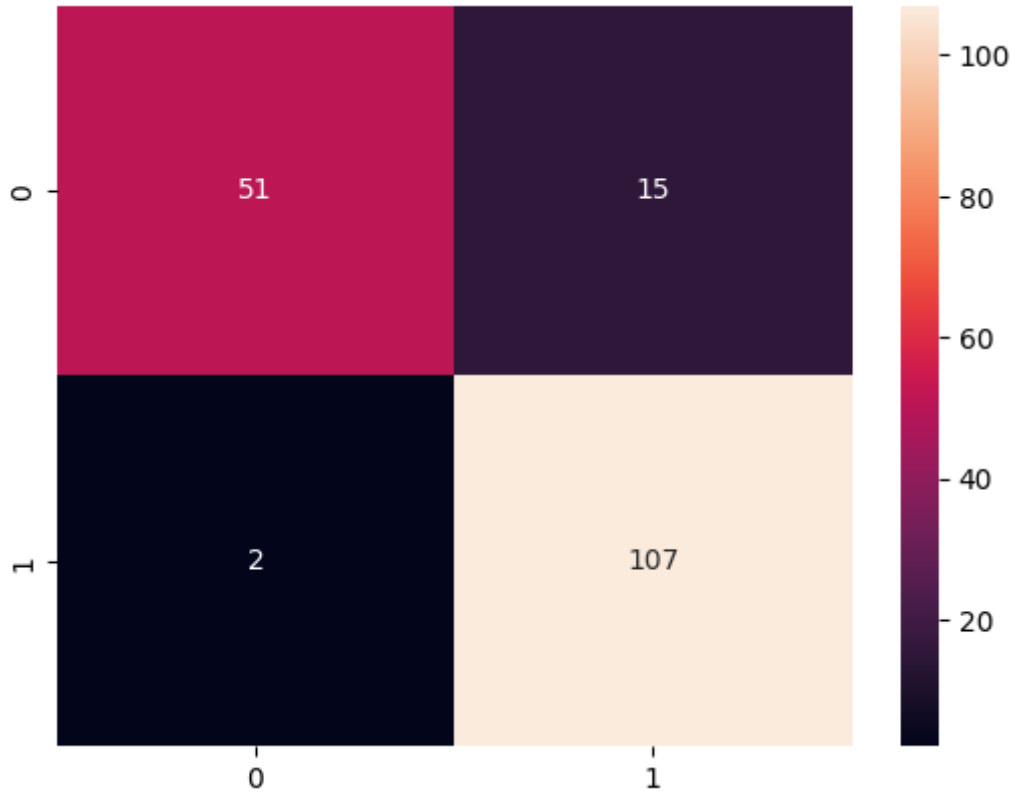
The SVM classifier provides a maximum accuracy (~95%) for the Gaussian kernel with a Test Size of 40%.

After applying PCA, the accuracy increased to ~96%.

In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier")
performOperationPCA(X_pca, y, model=MLPClassifier(max_iter=1000, learning_rate='adaptive'), test_split=0.5, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       0.96      0.77      0.86        66
           1       0.88      0.98      0.93       109

    accuracy                           0.90       175
   macro avg       0.92      0.88      0.89       175
weighted avg       0.91      0.90      0.90       175

Confusion Matrix:
```



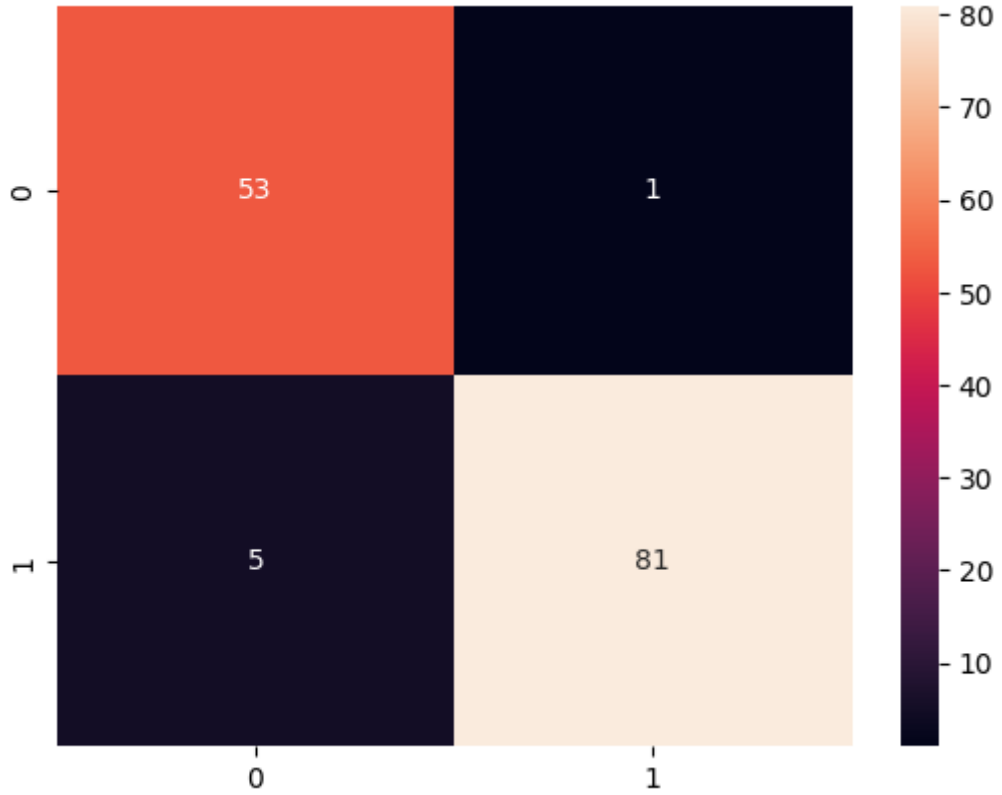
The MLP classifier provides a maximum accuracy (~91%) with a Test Size of 50%.

After applying PCA, the accuracy remains same.

In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier")
performOperationPCA(X_pca, y, model=RandomForestClassifier(), test_split=0.4, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier
Performance Evolution:
              precision    recall  f1-score   support

           0       0.91      0.98      0.95        54
           1       0.99      0.94      0.96        86

    accuracy                           0.96       140
   macro avg       0.95      0.96      0.96       140
weighted avg       0.96      0.96      0.96       140

Confusion Matrix:
```



The Random Forest classifier achieves a peak accuracy of approximately 96% at a test size of 40%.

However, the accuracy drops to around 94% after the application of PCA.

In [ ]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
```

In [ ]:
```
!gdown 1E-nGzdQpuaTEJerqFxPNvlTyC0zXVXgH
```

```
Downloading...
From: https://drive.google.com/uc?id=1E-nGzdQpuaTEJerqFxPNvlTyC0zXVXgH
To: /content/iris.data
100% 4.55k/4.55k [00:00<00:00, 16.2MB/s]
```

In [ ]:
```
cols = ["sepal_length", "septal_width", 'petal_length', 'petal_width', 'species']
df = pd.read_csv("iris.data", names=cols)
df.head()
```

Out[ ]:

	sepal_length	septal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

In [ ]:
```
df.drop_duplicates(subset=None, keep='first', inplace=True)
```

In [ ]:
```
X = df.drop('species', axis=1)
y = df['species']
```

In [ ]:

```
# Function for splitting, scaling, and sampling the dataset
def preprocessing(X, y, scaler, sampler, test_split):
  if(scaler != None):
    X = scaler.fit_transform(X)
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_split, random_state=10)
  if(sampler != None):
    X_train, y_train = RandomOverSampler().fit_resample(X_train, y_train)
  return X_train, y_train, X_test, y_test
```

In [ ]:

```
# Function for training and prediction
def trainAndPredict(model, X_train, y_train, X_test, y_test):
  model = model.fit(X_train, y_train)
  y_pred = model.predict(X_test)
  accuracy = accuracy_score(y_test, y_pred)
  report = classification_report(y_test, y_pred)
  conf_matrix = confusion_matrix(y_test, y_pred)
  return accuracy, report, conf_matrix
```

In [ ]:

```
# Function for generating an image illustrating Receiver Operating Characteristic (ROC) curve and Area Under the Curve (AUC)
def plot_roc_auc_multiclass(X, y, test_size, model, sampler, scaler):
  X_train, y_train, X_test, y_test = preprocessing(X, y, sampler=sampler, scaler=scaler, test_split=test_size)
  model.fit(X_train, y_train)
  scores = None
  if hasattr(model, "predict_proba") :
    scores = model.predict_proba(X_test)
  else :
    scores = model.decision_function(X_test)

  classes = y.unique()
  n_classes = len(classes)
  y_true_bin = label_binarize(y_test, classes=classes)

  fpr = dict()
  tpr = dict()
  roc_auc = dict()

  for i in range(n_classes):
      fpr[i], tpr[i], _ = roc_curve(y_true_bin[:, i], scores[:, i])
      roc_auc[i] = auc(fpr[i], tpr[i])

  plt.figure(figsize=(8, 6))
  for i in range(n_classes):
      plt.plot(fpr[i], tpr[i], label=f'Class {classes[i]} (AUC = {roc_auc[i]:.2f})')
```

```
  plt.plot([0, 1], [0, 1], color='gray', 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('ROC Curve for Multiclass Classification')
  plt.legend(loc='lower right')
  plt.show()
```

In [ ]:

```
def performOperation(X, y, model, title, scaler=None, sampler=None):
  print("\n\n**********************************************************************************************************************")
  print(title);
  graph_x = []
  graph_y = []
  max_accuracy = 0
  test_size = 0
  final_report = None
  final_conf_matrix = None
  for test_split in range(3, 8):
    X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=scaler, sampler=sampler,  test_split=test_split*0.1)
    accuracy, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
    if max_accuracy < accuracy:
      test_size = test_split*0.1
      max_accuracy = accuracy
      final_report = report
      final_conf_matrix = conf_matrix
    graph_x.append(test_split*10)
    graph_y.append(accuracy*100)
  print("Got Maximum Accuracy for Test Size = ", int(test_size*100), "%")
  print("Maximum Accuracy = ", max_accuracy*100, "%")
  print("Performance Evolution:")
  print(final_report)
  print("Confusion Matrix:")
  sns.heatmap(final_conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
  plt.plot(graph_x, graph_y, marker='o')
  plt.xlabel('Test Size')
  plt.ylabel('Accuracy')
  plt.title(title)
  plt.show()
  print("\n")
  plot_roc_auc_multiclass(X, y, test_size=test_size, model=model, scaler=scaler, sampler=sampler)
  print("\n\n**********************************************************************************************************************")
  return graph_y
```

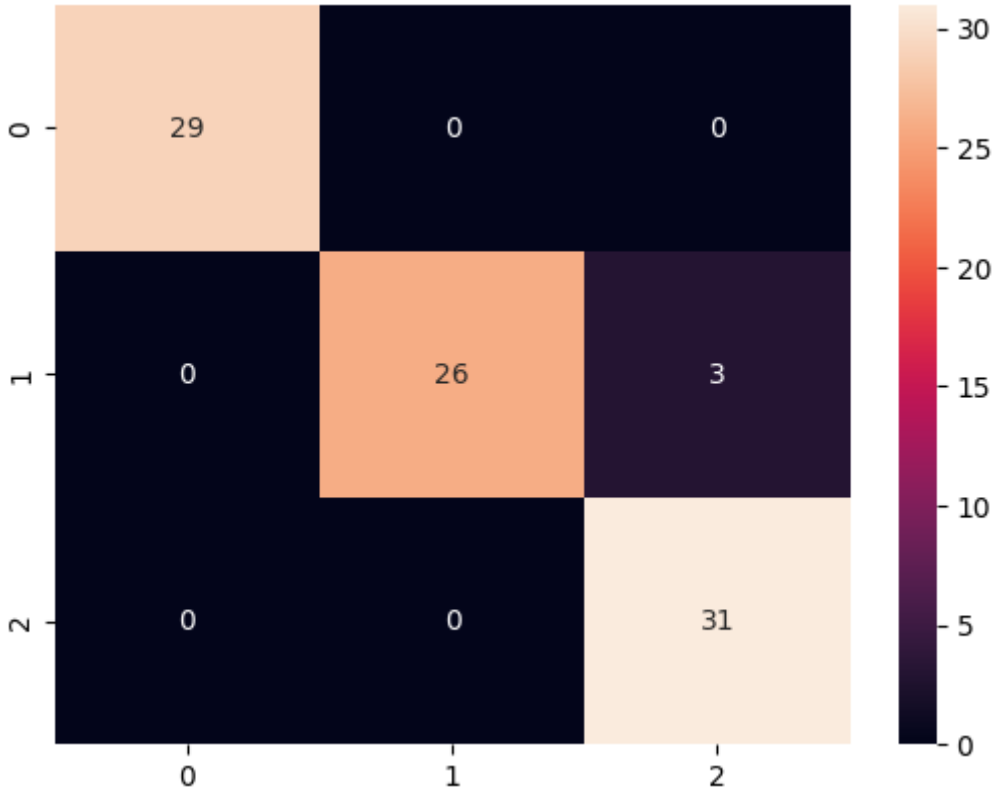
In [ ]:

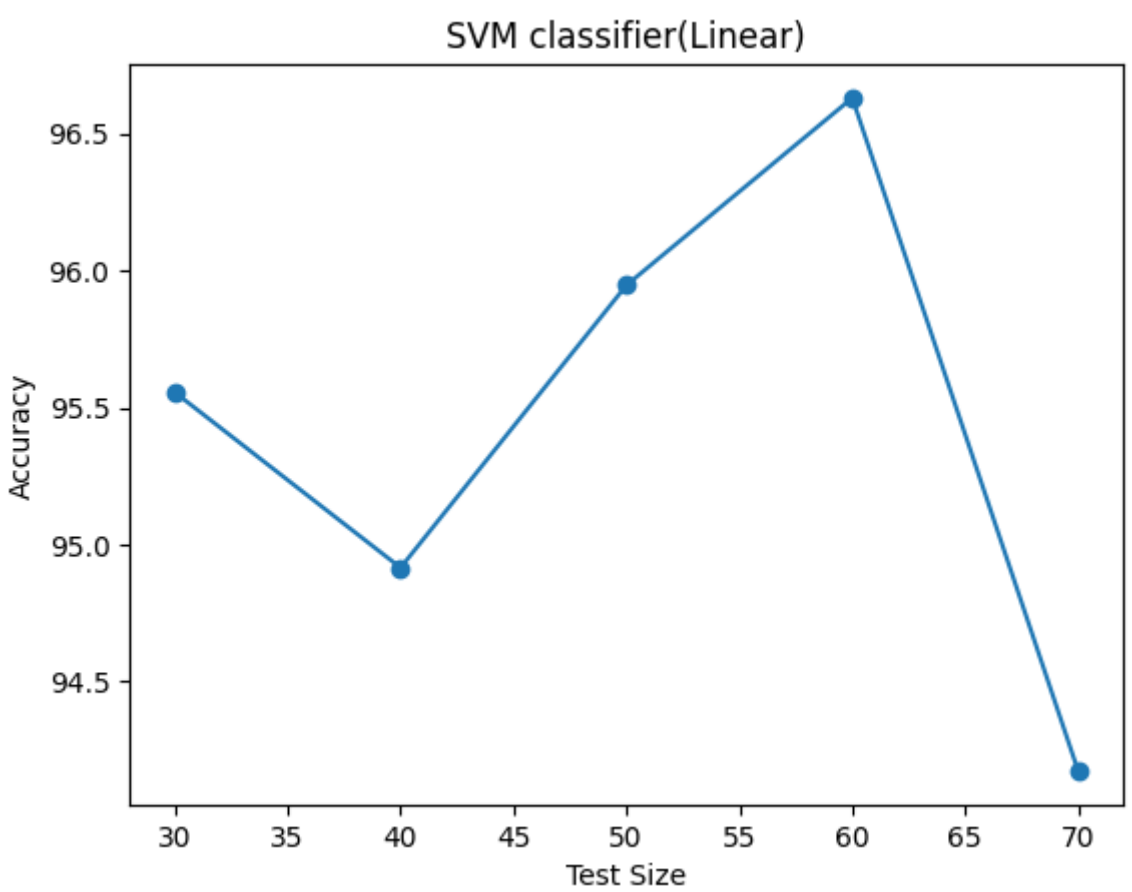
```
def performOperationPCA(X, y, model, test_split, sampler=None, scaler=None):
  X_train, y_train, X_test, y_test = preprocessing(X, y, scaler=None, sampler=sampler,  test_split=test_split)
  _, report, conf_matrix = trainAndPredict(model, X_train, y_train, X_test, y_test)
  print("Performance Evolution:")
  print(report)
  print("Confusion Matrix:")
  sns.heatmap(conf_matrix, annot=True, fmt='d')
  plt.show()
  print("\n")
```

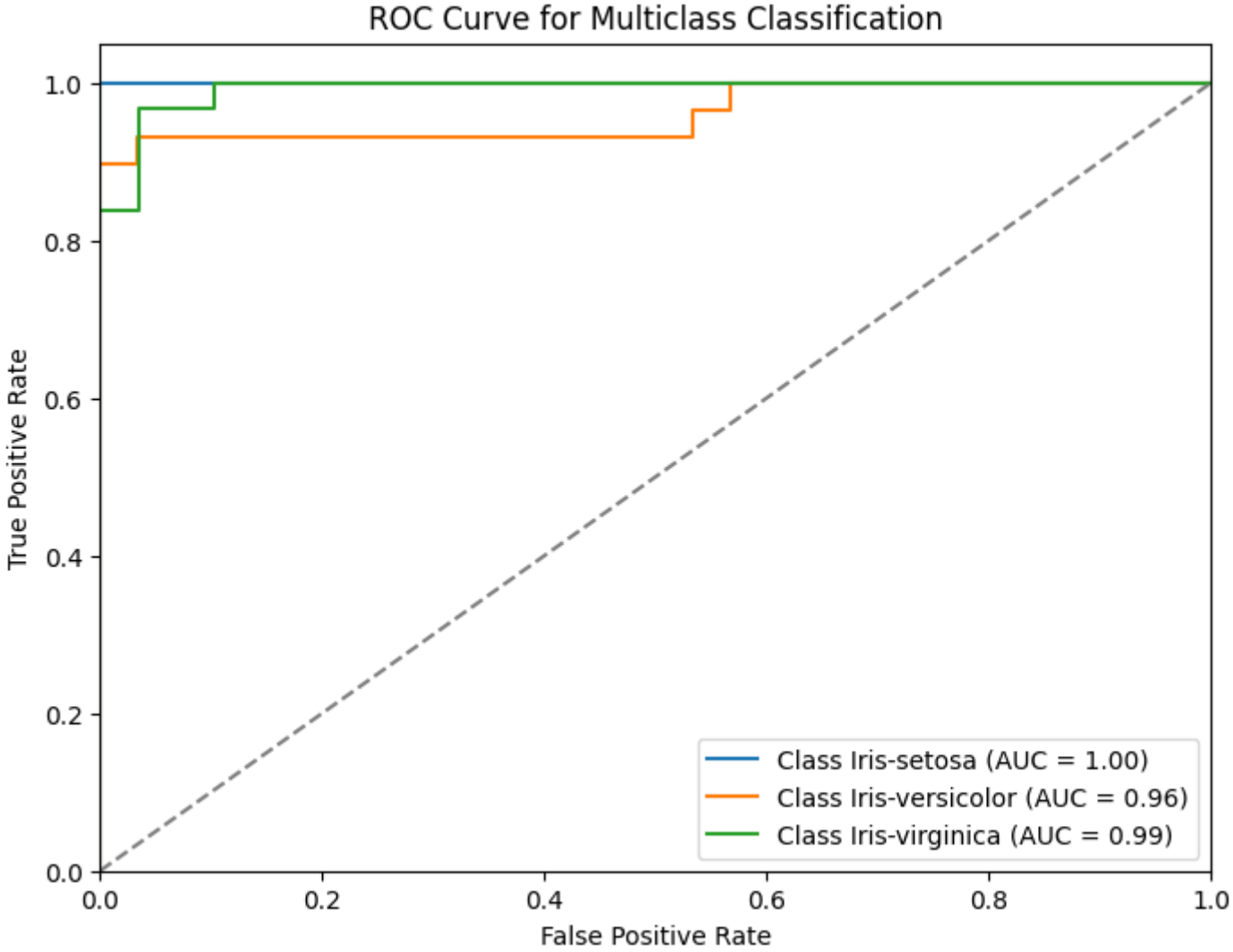
In [ ]:

```
svm_data = []
svm_data.append(performOperation(X, y, model=SVC(kernel='linear'), title="SVM classifier(Linear)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='poly'), title="SVM classifier(Polynomial)", sampler=RandomOverSampler()))
svm_data.append(performOperation(X, y, model=SVC(kernel='sigmoid', gamma=0.1), title="SVM classifier(Sigmoid)", sampler=RandomOverSampler(),
svm_data.append(performOperation(X, y, model=SVC(kernel='rbf'), title="SVM classifier(Gaussian)", sampler=RandomOverSampler()))
```

```
**************************************************************************************************************************
**************************************************


SVM classifier(Linear)
Got Maximum Accuracy for Test Size =  60 %
Maximum Accuracy =  96.62921348314607 %
Performance Evolution:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        29
Iris-versicolor       1.00      0.90      0.95        29
 Iris-virginica       0.91      1.00      0.95        31

       accuracy                           0.97        89
      macro avg       0.97      0.97      0.97        89
   weighted avg       0.97      0.97      0.97        89

Confusion Matrix:
```


29
0
0
0
26
3
0
0
31
0
1
2
0
1
2
30
25
20
15
10
5
0


SVM classifier(Linear)
96.5
96.0
95.5
95.0
94.5
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

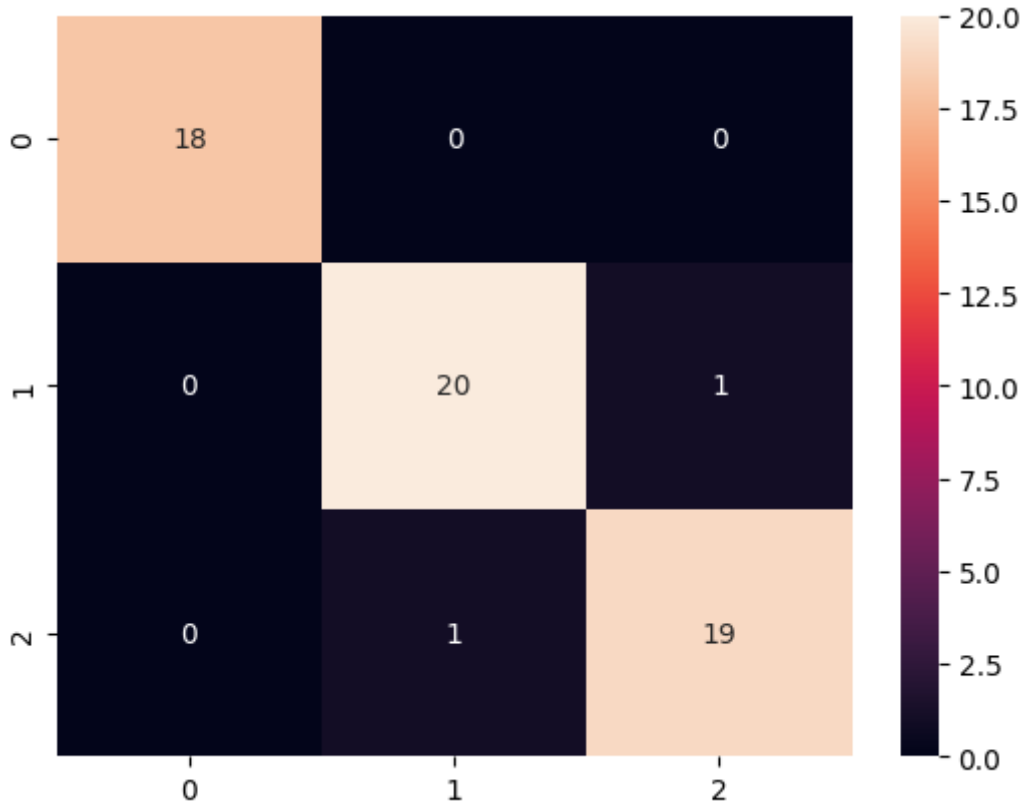
****

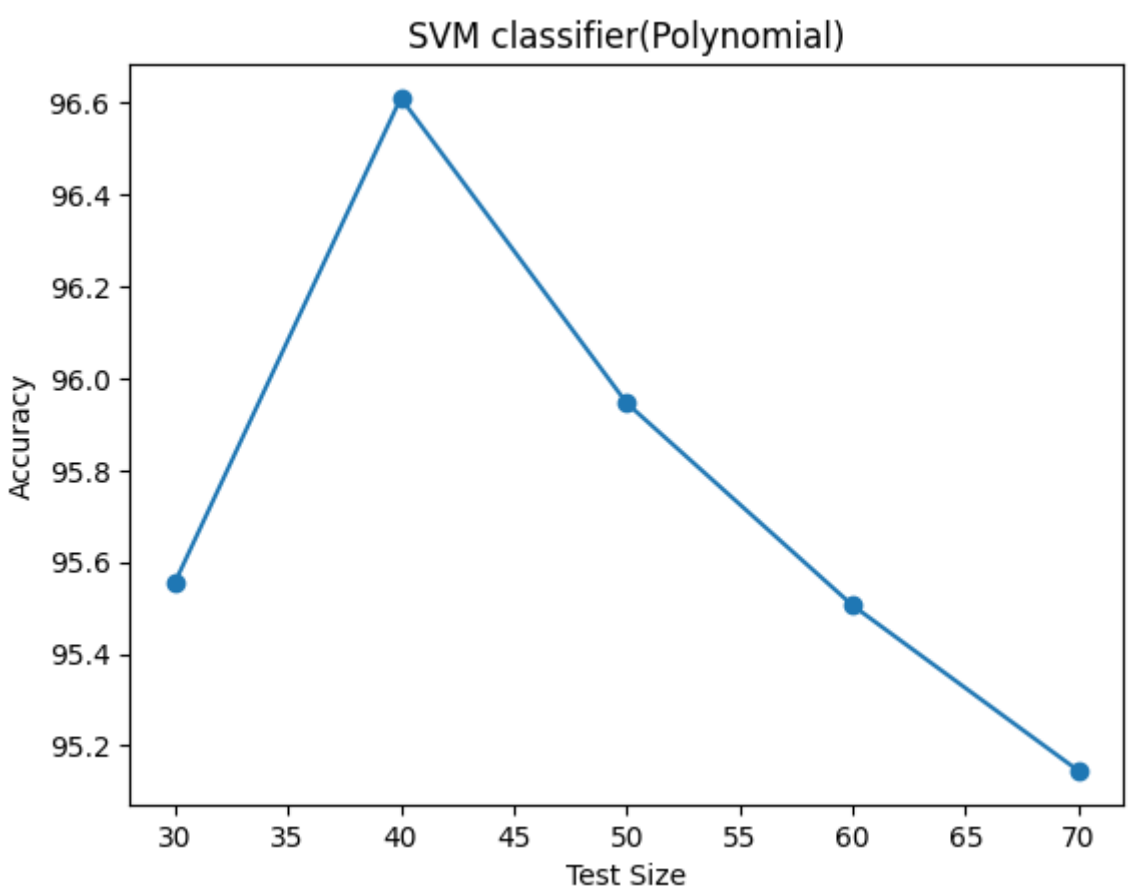
SVM classifier(Polynomial)
Got Maximum Accuracy for Test Size = 40 %
Maximum Accuracy = 96.61016949152543 %

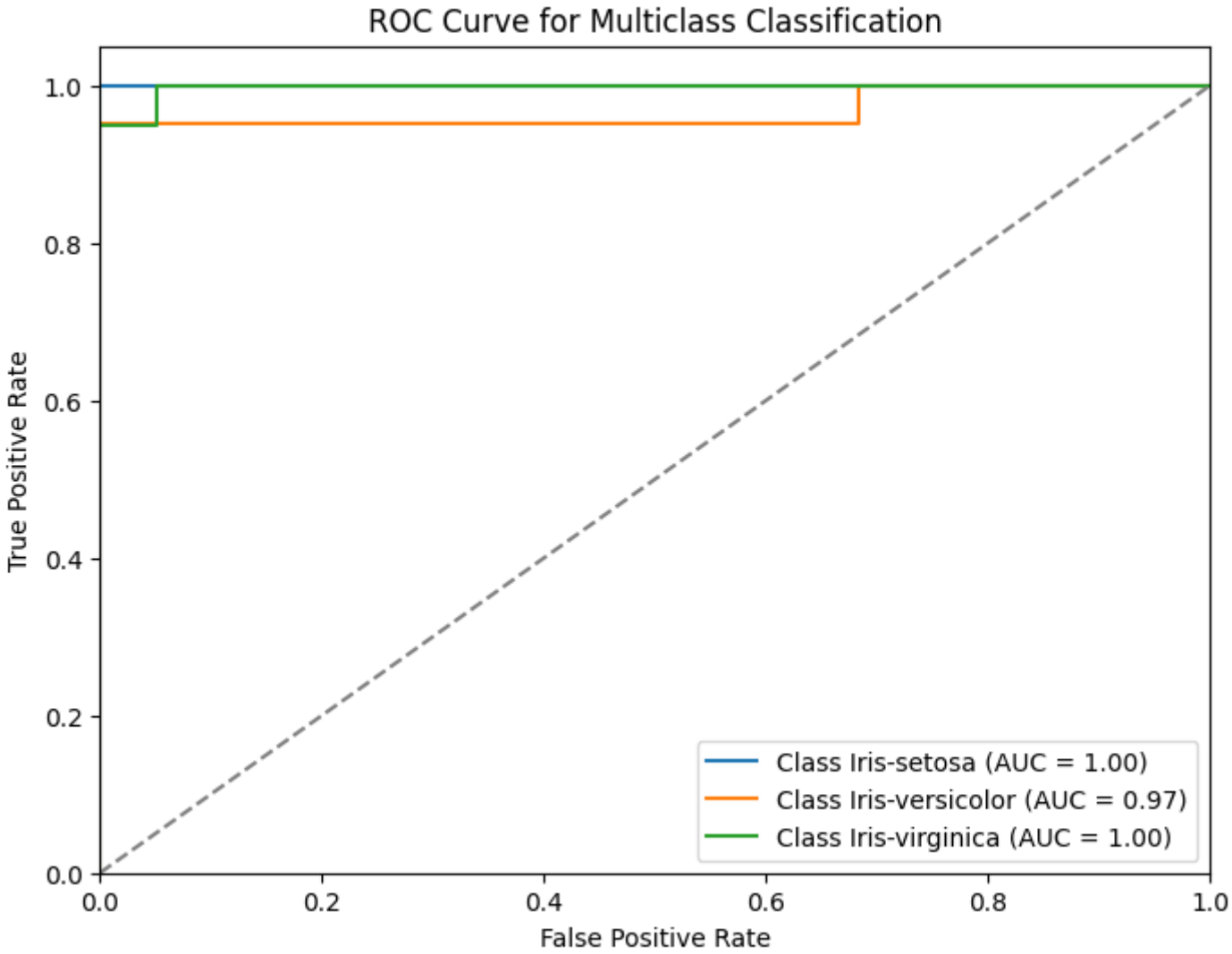
Performance Evolution:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	18
Iris-versicolor	0.95	0.95	0.95	21
Iris-virginica	0.95	0.95	0.95	20
accuracy			0.97	59
macro avg	0.97	0.97	0.97	59
weighted avg	0.97	0.97	0.97	59

Confusion Matrix:




SVM classifier(Polynomial)
96.6
96.4
96.2
96.0
95.8
95.6
95.4
95.2
Accuracy
30
35
40
45
50
55
60
65
70
Test Size



****

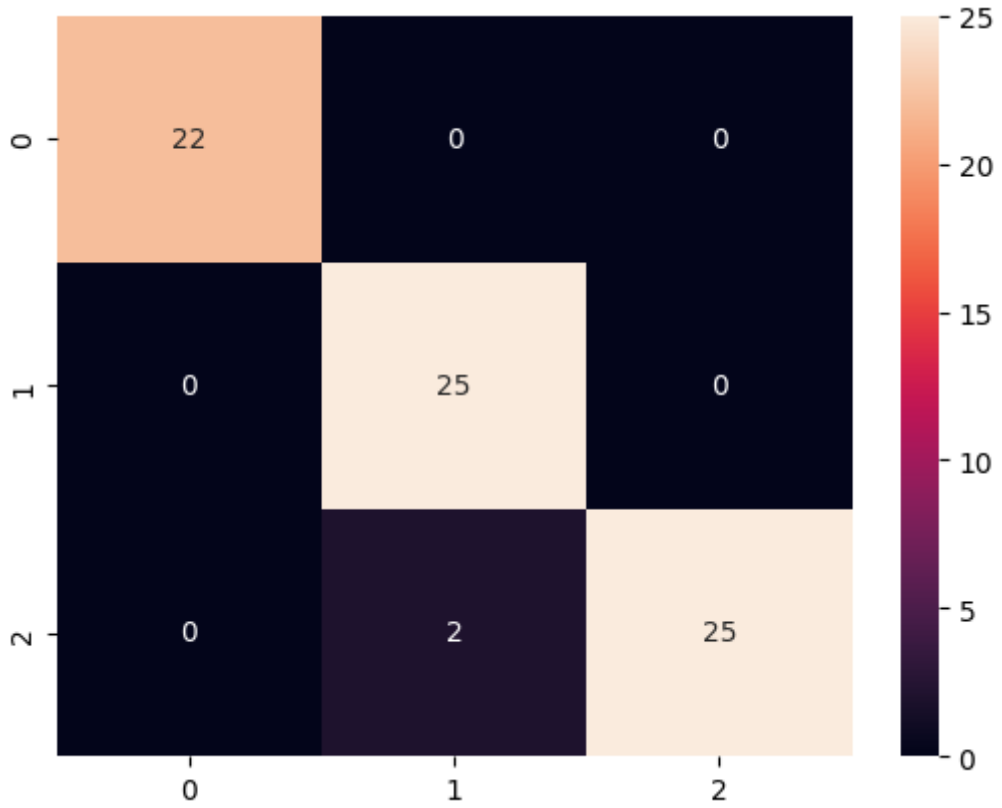
****

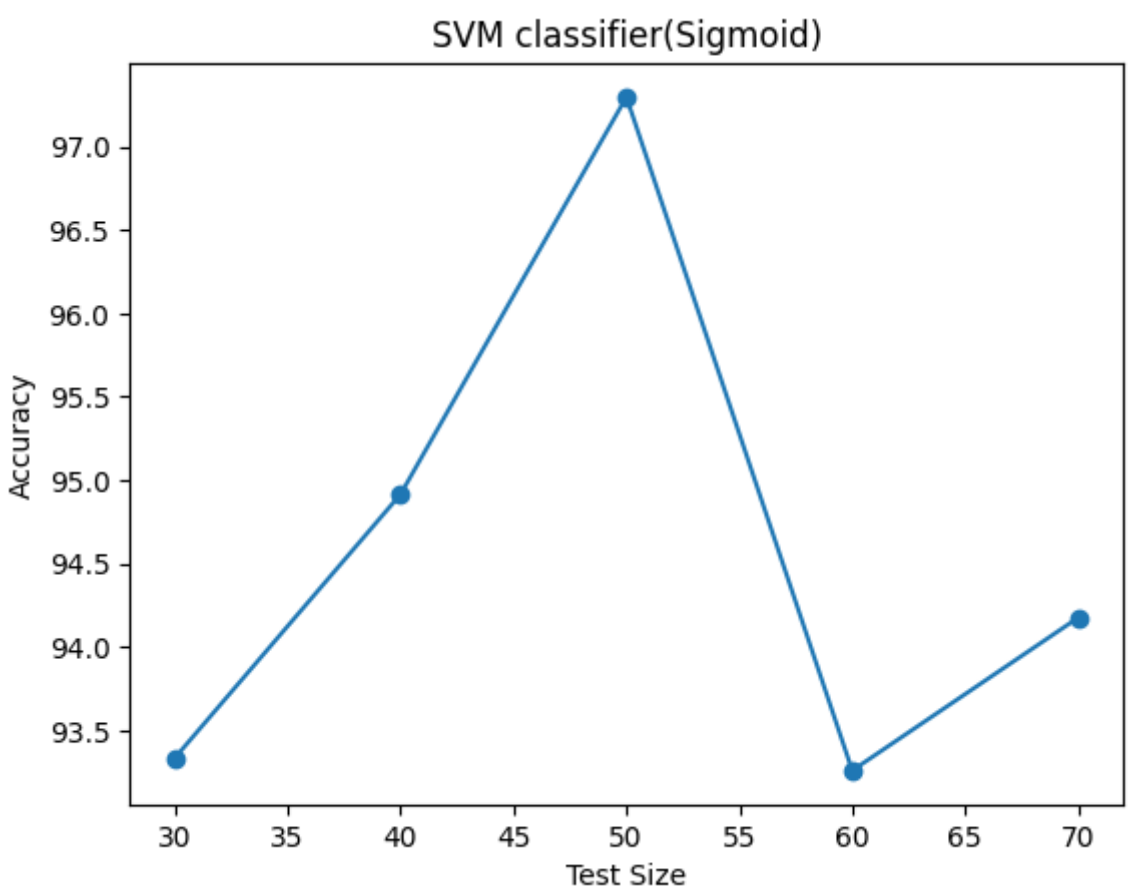
SVM classifier(Sigmoid)
Got Maximum Accuracy for Test Size = 50 %
Maximum Accuracy = 97.2972972972973 %

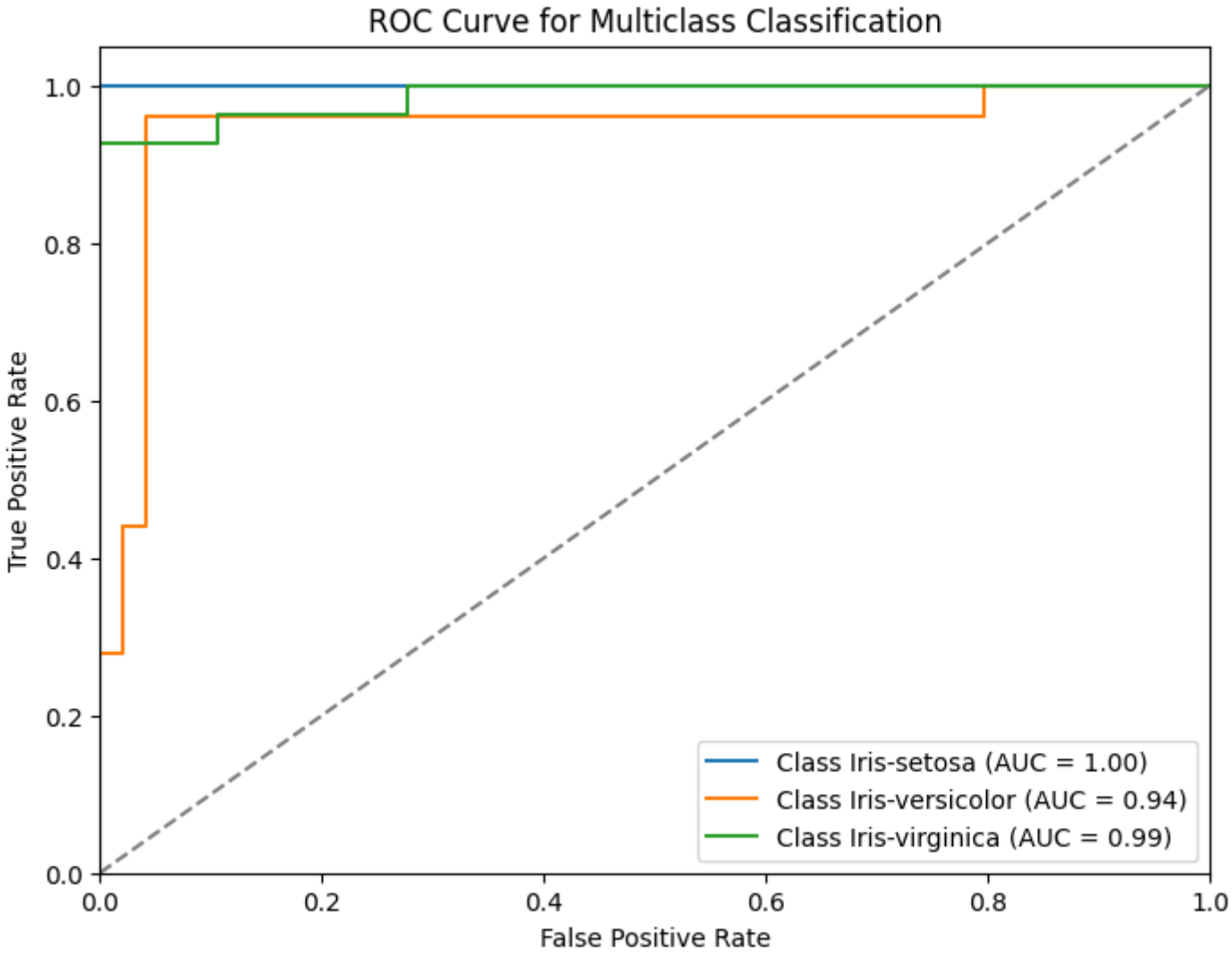
Performance Evolution:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	22
Iris-versicolor	0.93	1.00	0.96	25
Iris-virginica	1.00	0.93	0.96	27
accuracy			0.97	74
macro avg	0.98	0.98	0.97	74
weighted avg	0.97	0.97	0.97	74

Confusion Matrix:




SVM classifier(Sigmoid)
Accuracy
97.0
96.5
96.0
95.5
95.0
94.5
94.0
93.5
30
35
40
45
50
55
60
65
70
Test Size



****

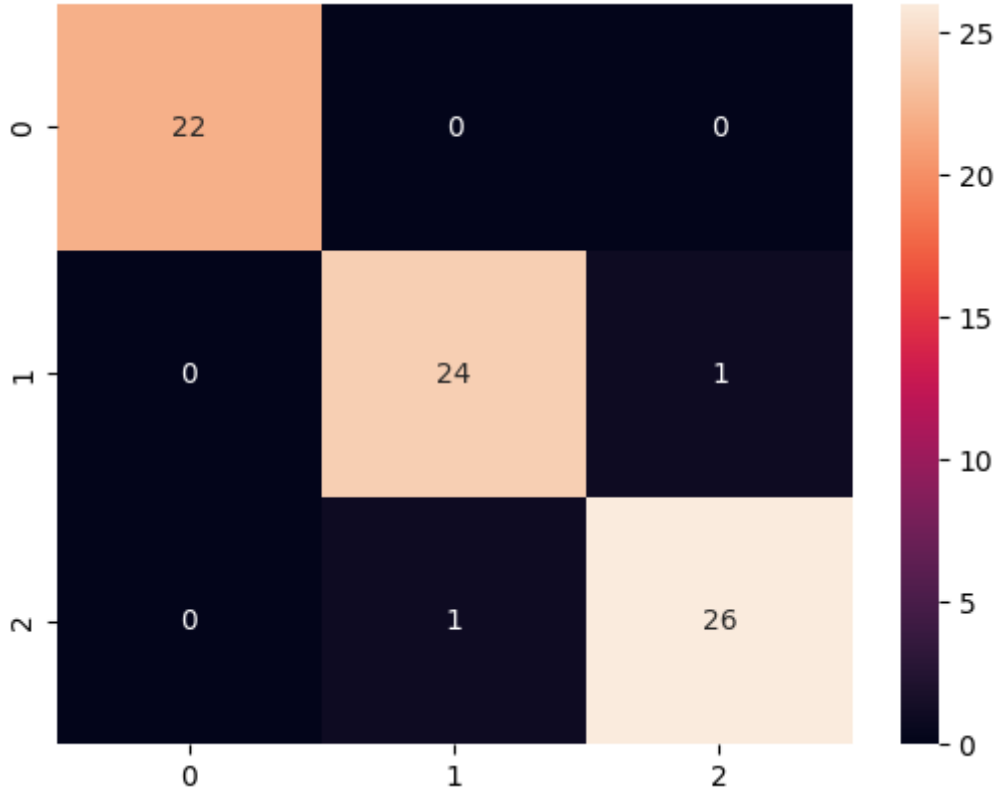
****

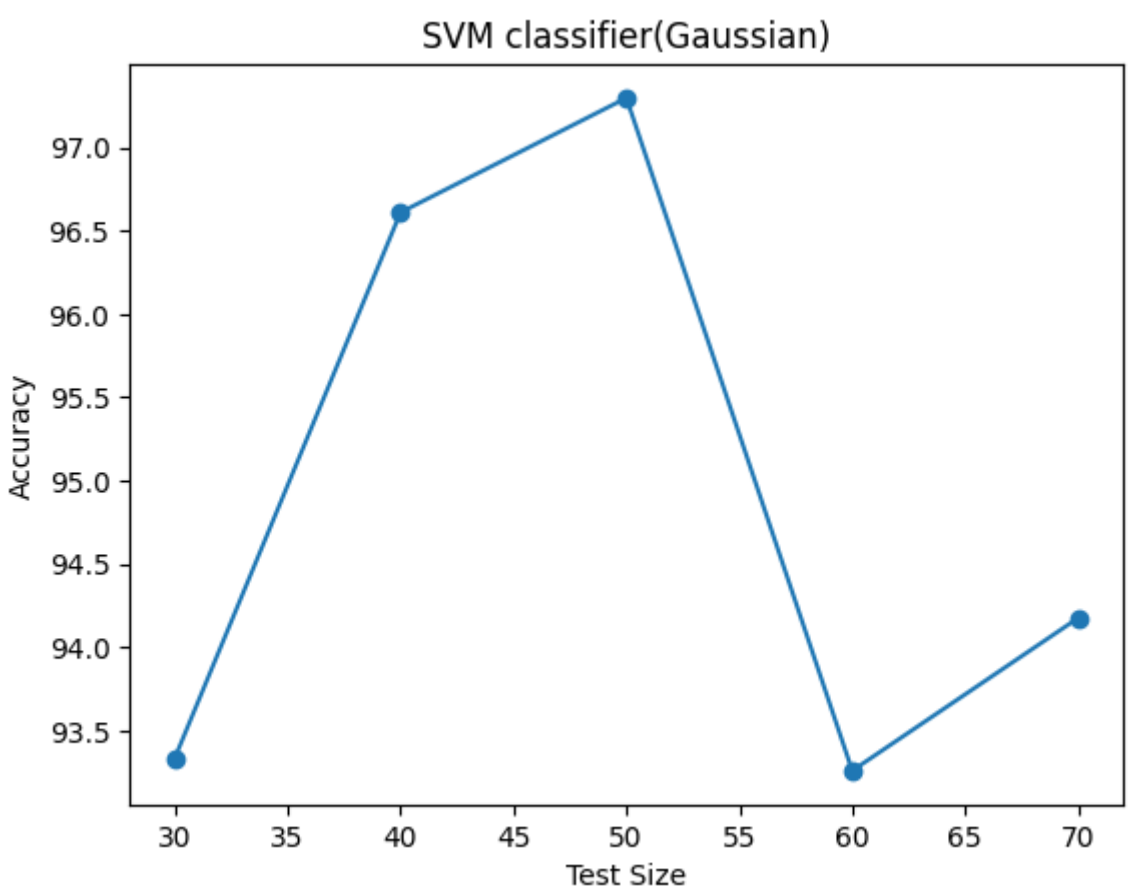
SVM classifier(Gaussian)
Got Maximum Accuracy for Test Size = 50 %
Maximum Accuracy = 97.2972972972973 %

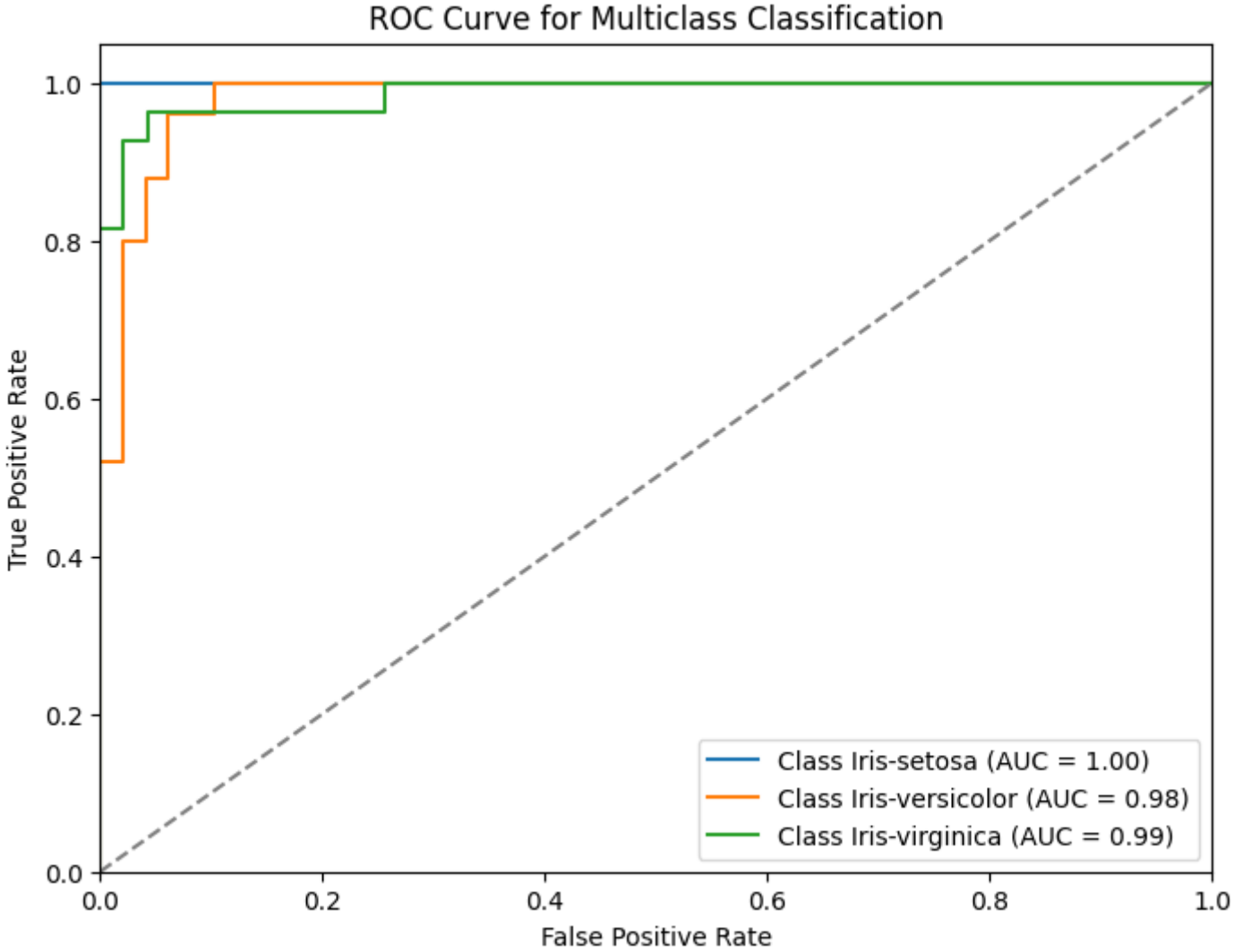
Performance Evolution:

	precision	recall	f1-score	support
Iris-setosa	1.00	1.00	1.00	22
Iris-versicolor	0.96	0.96	0.96	25
Iris-virginica	0.96	0.96	0.96	27
accuracy			0.97	74
macro avg	0.97	0.97	0.97	74
weighted avg	0.97	0.97	0.97	74

Confusion Matrix:




SVM classifier(Gaussian)
Accuracy
97.0
96.5
96.0
95.5
95.0
94.5
94.0
93.5
30
35
40
45
50
55
60
65
70
Test Size



****

In [ ]:

```
print("Table showing accuracy for different SVM kernels at various test sizes\n")
table = pd.DataFrame(svm_data, index=["SVM (Linear)", 'SVM (Polynomial)', 'SVM (Sigmoid)', 'SVM (Gaussian)'], columns = [f"Test Size = {10 *
print(table)
```

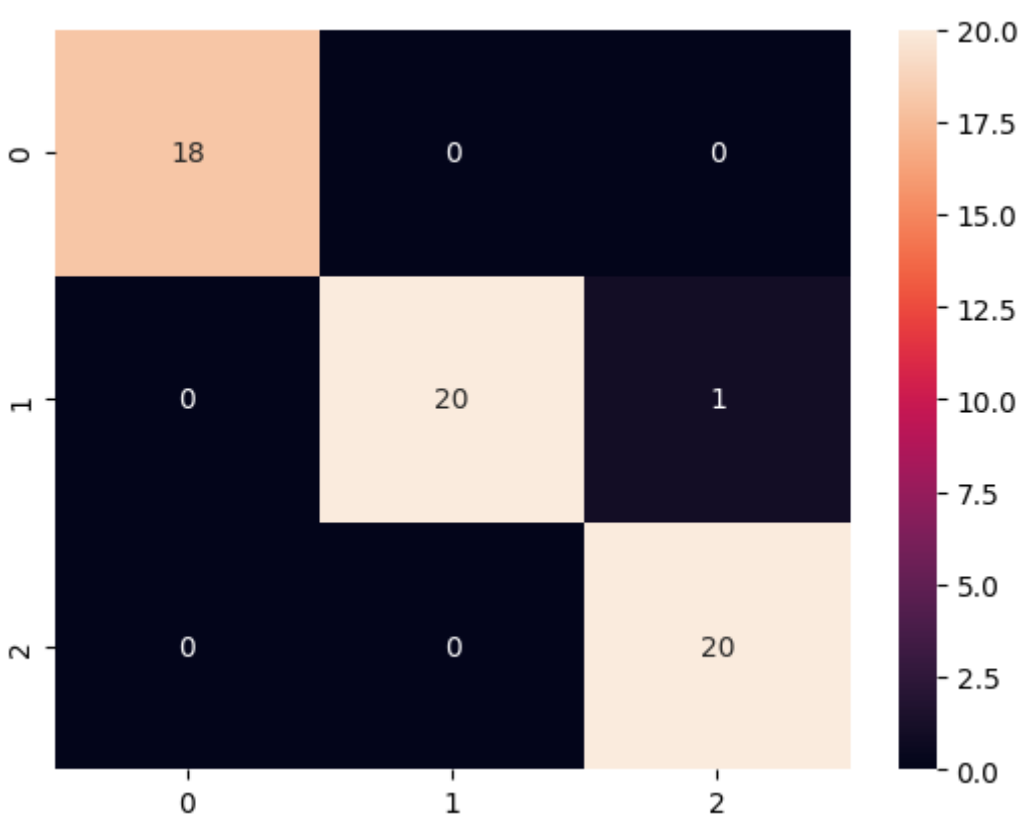
```
Table showing accuracy for different SVM kernels at various test sizes

                 Test Size = 30%  Test Size = 40%  Test Size = 50%  \
SVM (Linear)           95.555556        94.915254        95.945946
```

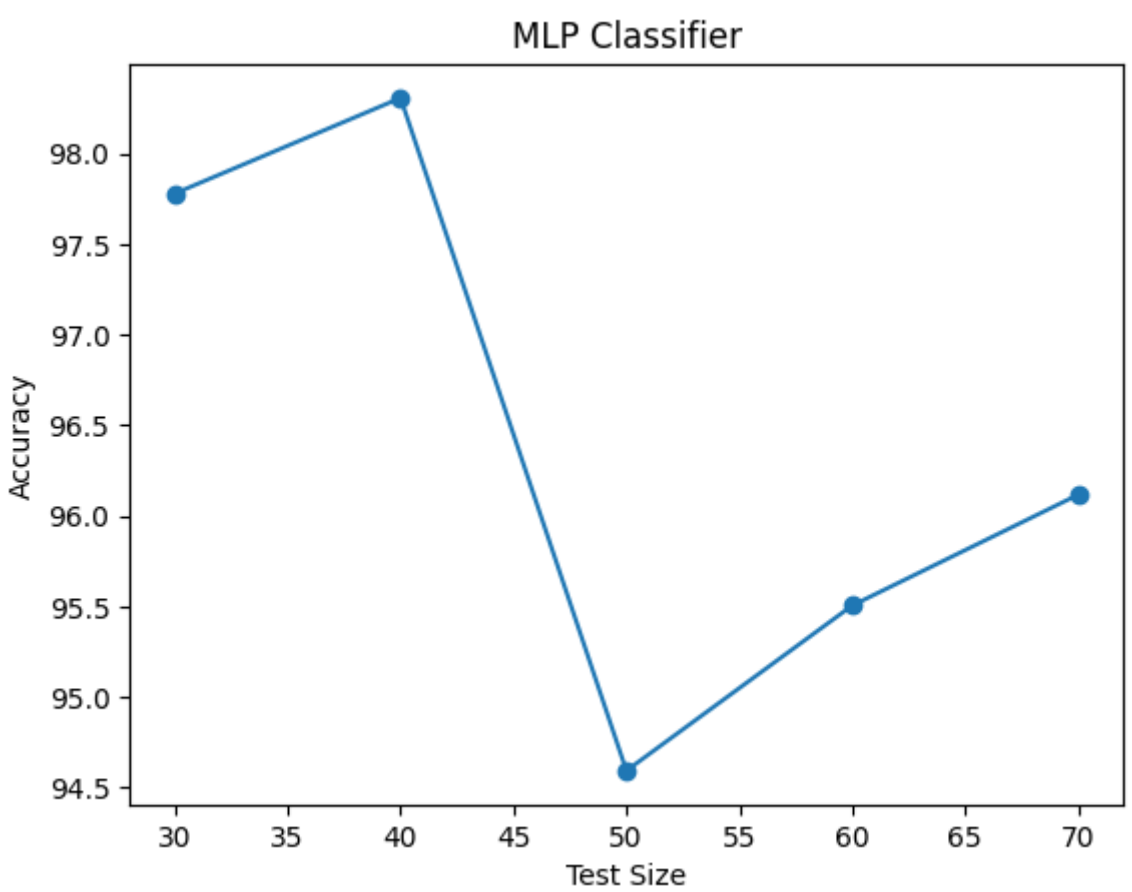
```
SVM (Polynomial)        95.555556        96.610169        95.945946
SVM (Sigmoid)           93.333333        94.915254        97.297297
SVM (Gaussian)          93.333333        96.610169        97.297297

                  Test Size = 60%  Test Size = 70%
SVM (Linear)            96.629213        94.174757
SVM (Polynomial)        95.505618        95.145631
SVM (Sigmoid)           93.258427        94.174757
SVM (Gaussian)          93.258427        94.174757
```

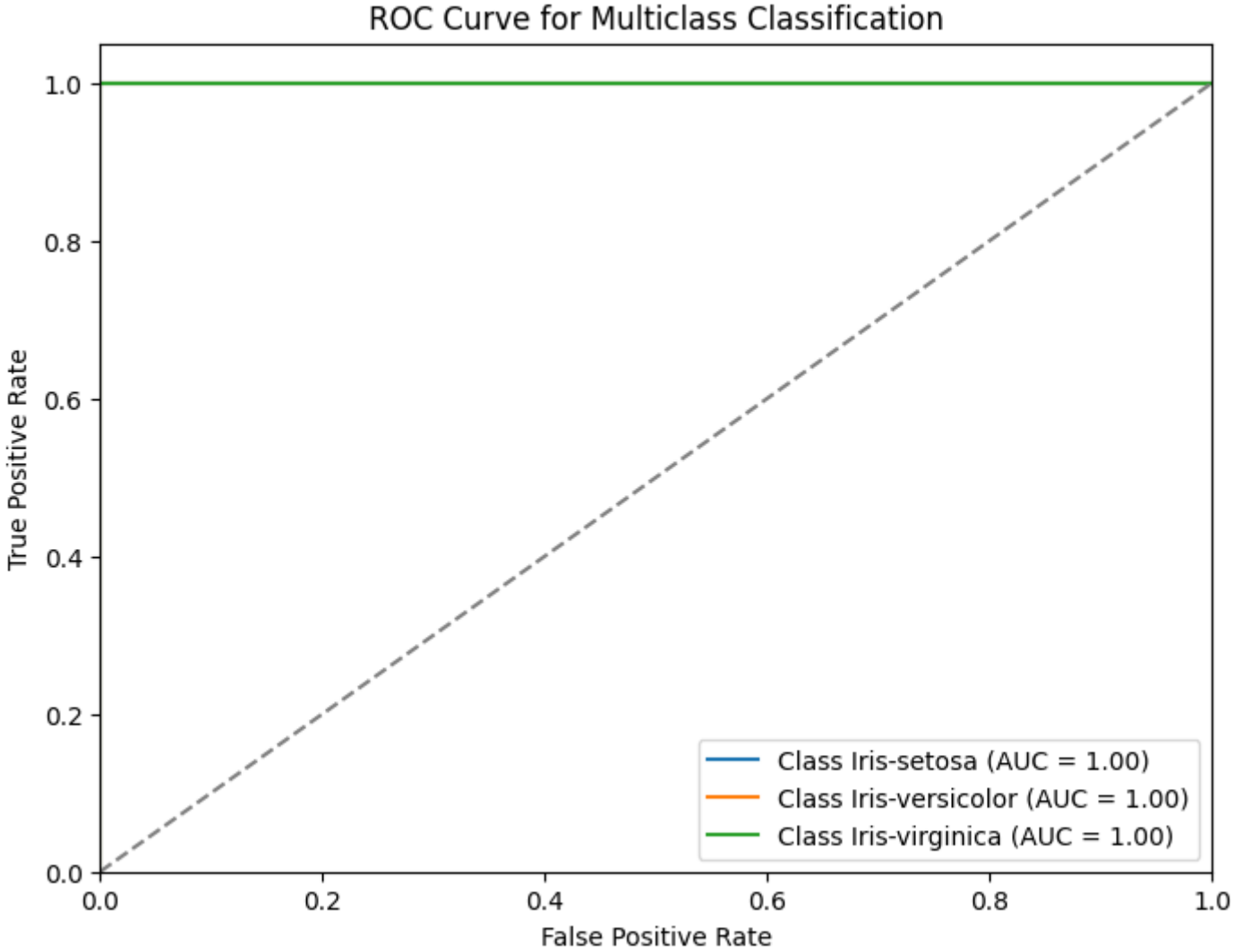
In [ ]:

```python
_ = performOperation(X, y, model=MLPClassifier(max_iter=1000), title="MLP Classifier", sampler=RandomOverSampler())
```

```
**************************************************************************************************************************************************************
**************************************************

MLP Classifier
Got Maximum Accuracy for Test Size =  40 %
Maximum Accuracy =  98.30508474576271 %
Performance Evolution:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       1.00      0.95      0.98        21
 Iris-virginica       0.95      1.00      0.98        20

       accuracy                           0.98        59
      macro avg       0.98      0.98      0.98        59
   weighted avg       0.98      0.98      0.98        59

Confusion Matrix:
```


18
0
0
0
20
1
0
0
20
0
1
2
0
1
2
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0


MLP Classifier
Accuracy
98.0
97.5
97.0
96.5
96.0
95.5
95.0
94.5
30
35
40
45
50
55
60
65
70
Test Size



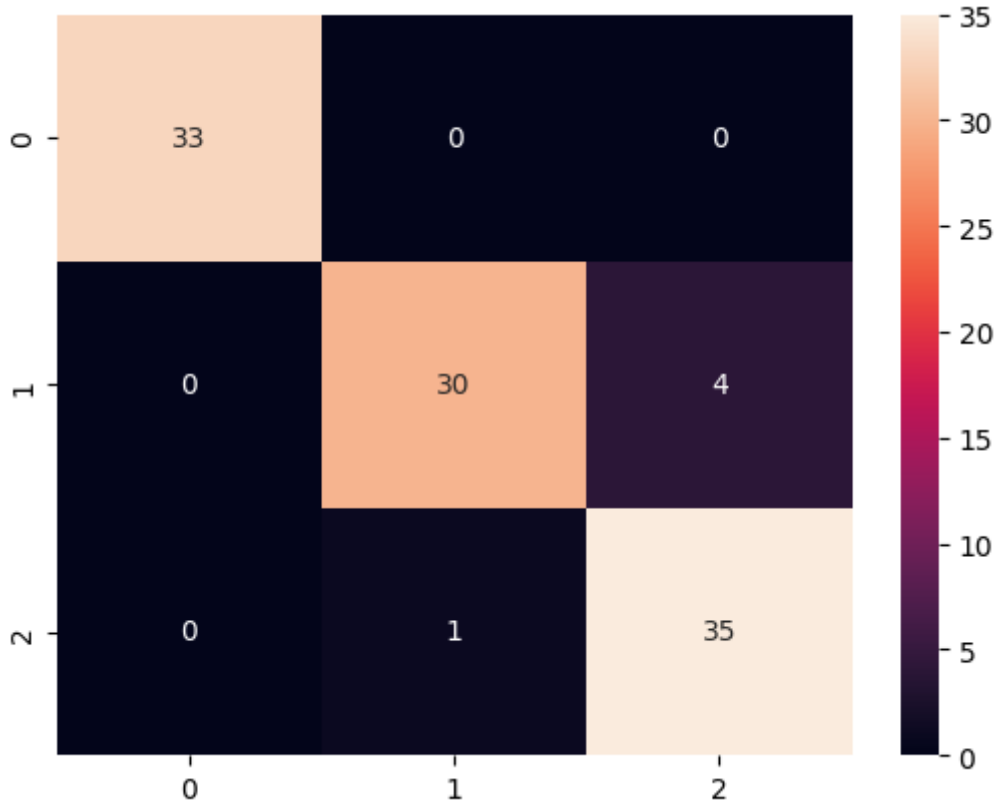
****

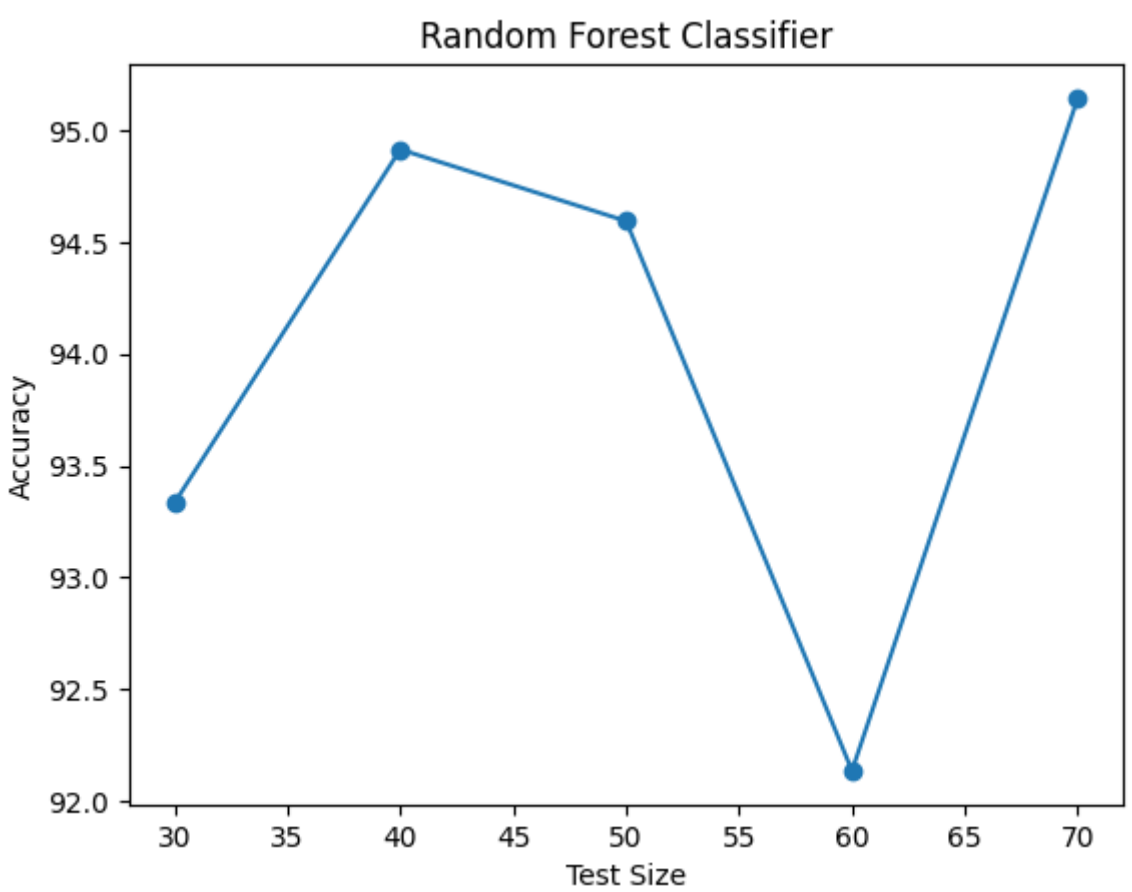
In [ ]:
```
_ = performOperation(X, y, model=RandomForestClassifier(), title="Random Forest Classifier", sampler=RandomOverSampler())
```

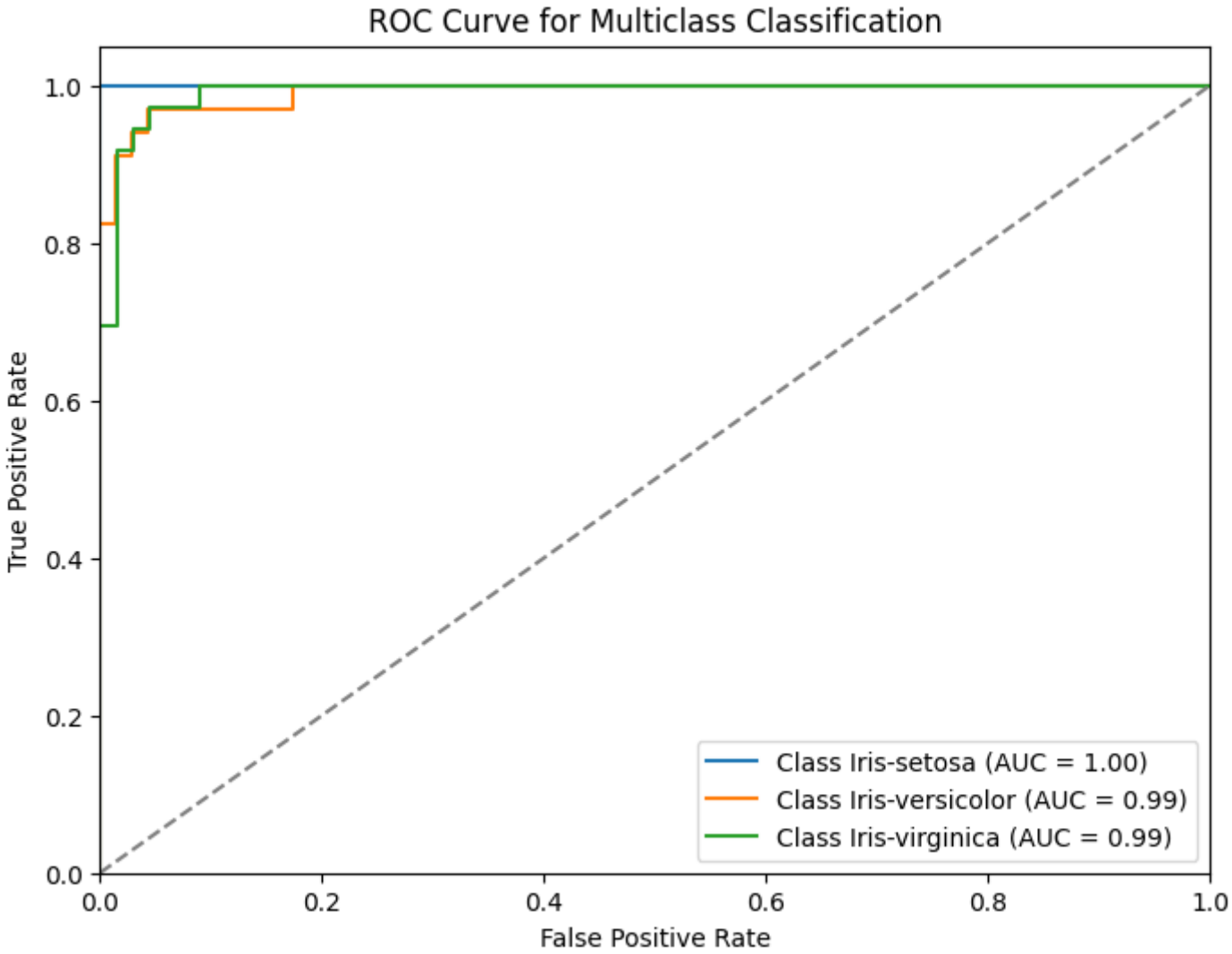
****

Random Forest Classifier

```
Got Maximum Accuracy for Test Size =  70 %
Maximum Accuracy =  95.14563106796116 %
Performance Evolution:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        33
Iris-versicolor       0.97      0.88      0.92        34
 Iris-virginica       0.90      0.97      0.93        36

       accuracy                           0.95       103
      macro avg       0.96      0.95      0.95       103
   weighted avg       0.95      0.95      0.95       103

Confusion Matrix:
```




Random Forest Classifier
Accuracy
95.0
94.5
94.0
93.5
93.0
92.5
92.0
30
35
40
45
50
55
60
65
70
Test Size



****

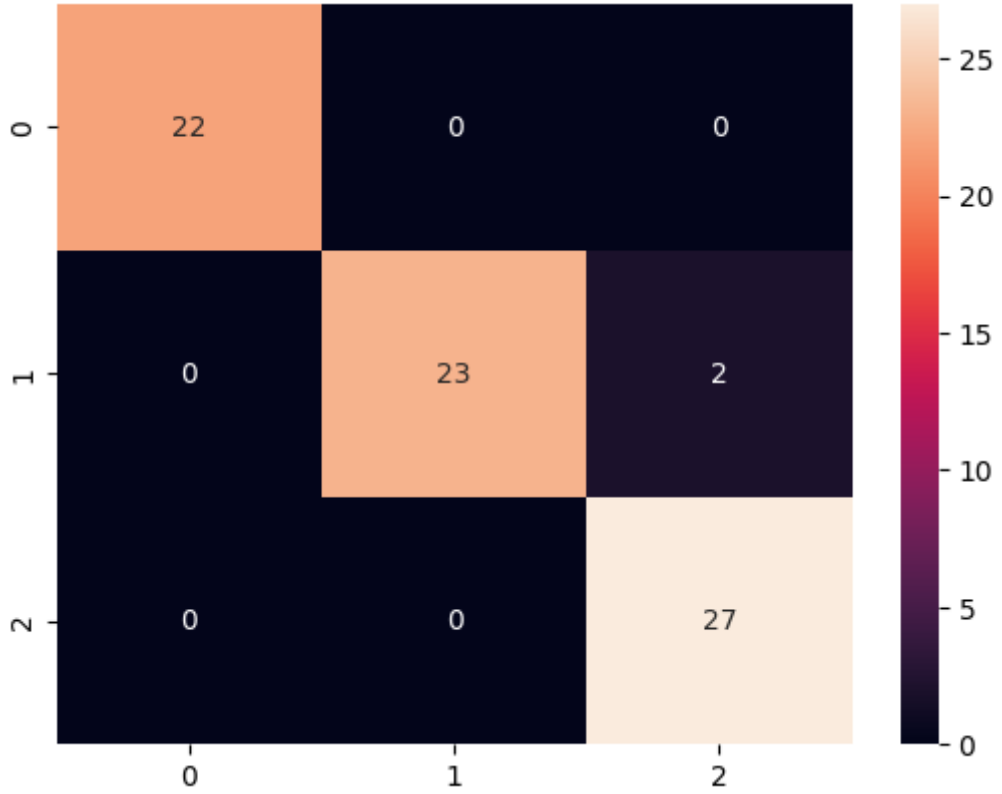
In [ ]:

```
# Applying Principal Component Analysis (PCA) for feature dimensionality reduction
pca = PCA(n_components=3)
X_pca = pca.fit_transform(X)
```

In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier")
performOperationPCA(X_pca, y, model=SVC(kernel='linear'), test_split=0.5, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> SVM classifier
Performance Evolution:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        22
Iris-versicolor       1.00      0.92      0.96        25
 Iris-virginica       0.93      1.00      0.96        27

       accuracy                           0.97        74
      macro avg       0.98      0.97      0.97        74
   weighted avg       0.97      0.97      0.97        74

Confusion Matrix:
```



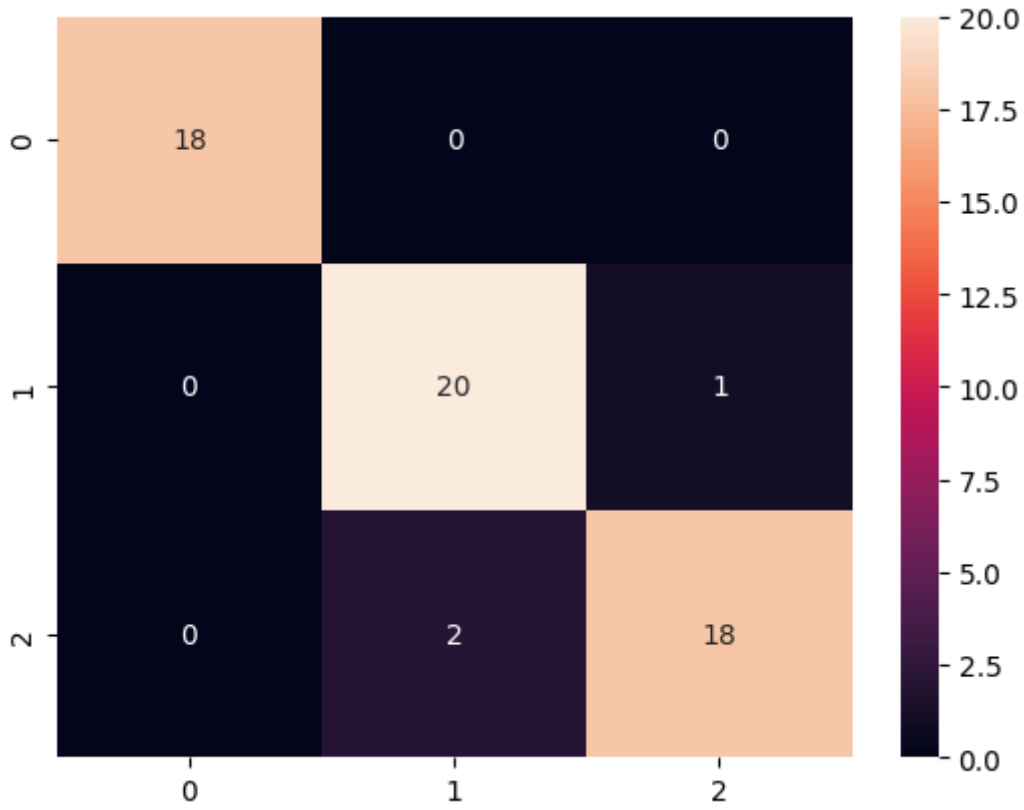
In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier")
performOperationPCA(X_pca, y, model=MLPClassifier(max_iter=1000), test_split=0.4, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> MLP classifier
Performance Evolution:
                precision    recall  f1-score   support
```

```
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       0.91      0.95      0.93        21
 Iris-virginica       0.95      0.90      0.92        20

       accuracy                           0.95        59
      macro avg       0.95      0.95      0.95        59
   weighted avg       0.95      0.95      0.95        59
```

Confusion Matrix:



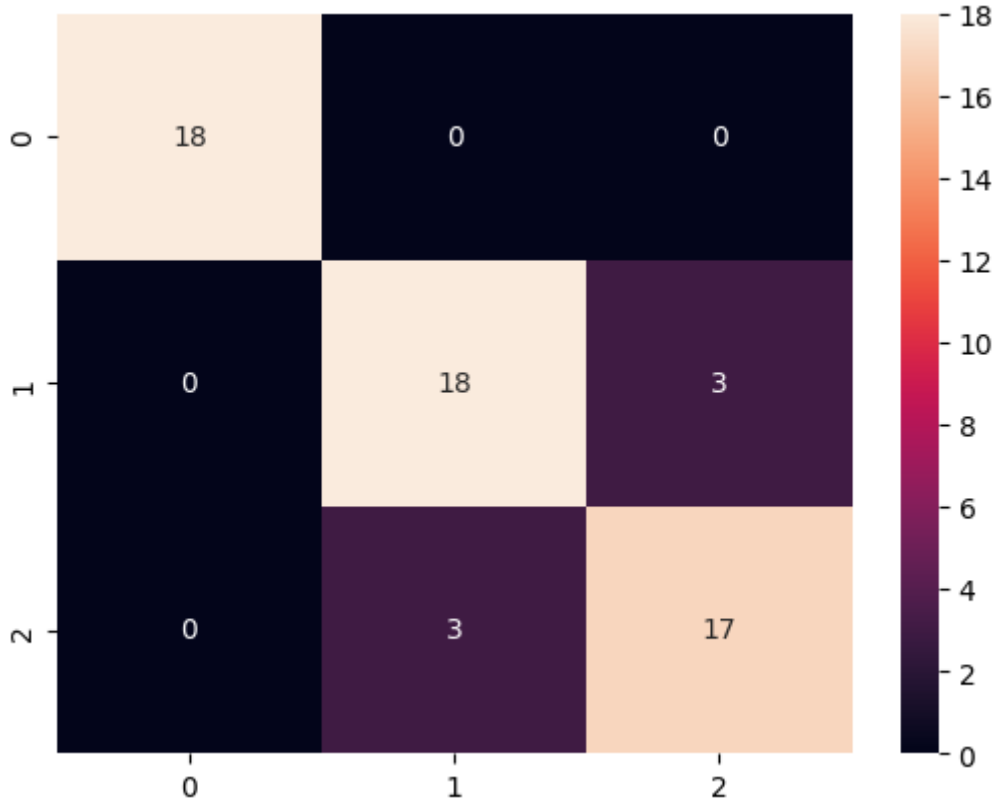
In [ ]:

```
print("After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier")
performOperationPCA(X_pca, y, model=RandomForestClassifier(), test_split=0.4, sampler=RandomOverSampler())
```

```
After Using Principal Component Analysis (PCA) for feature dimensionality reduction ---> Random Forest classifier
Performance Evolution:
                precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        18
Iris-versicolor       0.86      0.86      0.86        21
 Iris-virginica       0.85      0.85      0.85        20
```

accuracy			0.90	59
macro avg	0.90	0.90	0.90	59
weighted avg	0.90	0.90	0.90	59

Confusion Matrix:



The Random Forest classifier achieves a peak accuracy of approximately 95% at a test size of 40%.

However, the accuracy drops to around 93% after the application of PCA.