MACHINE LEARNING LABORATORY

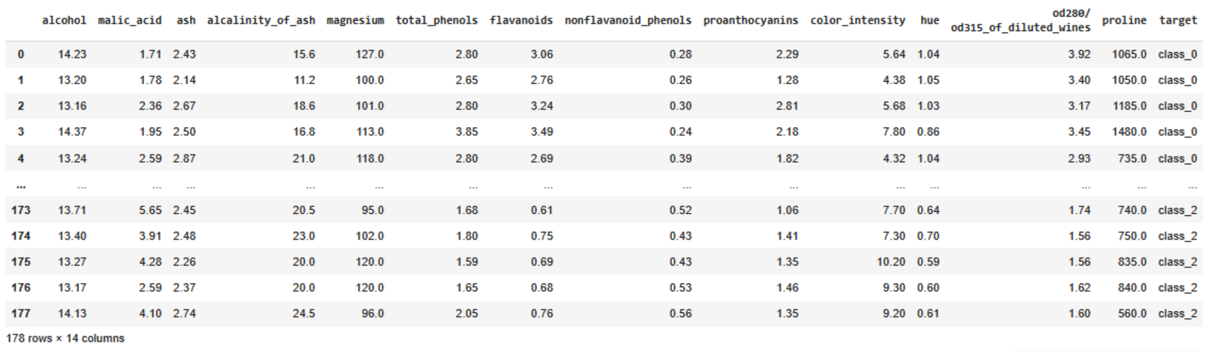
ASSIGNMENT 2
Name: Atmik Goswami
Roll No: 002211001104
Section: A2

Github: github.com/atmikgoswami/ML-Lab

DATASETS:

1. **Wine Dataset**
    - Features: 13 numeric features (e.g., alcohol, malic_acid, ash, etc.)
    - Classes: class_0, class_1, class_2
    - Total Samples: 178

Sample Data:

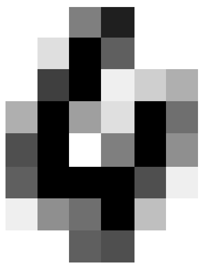


	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/ od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	class_0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	class_0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	class_0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	class_0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	class_0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
173	13.71	5.65	2.45	20.5	95.0	1.68	0.61	0.52	1.06	7.70	0.64	1.74	740.0	class_2
174	13.40	3.91	2.48	23.0	102.0	1.80	0.75	0.43	1.41	7.30	0.70	1.56	750.0	class_2
175	13.27	4.28	2.26	20.0	120.0	1.59	0.69	0.43	1.35	10.20	0.59	1.56	835.0	class_2
176	13.17	2.59	2.37	20.0	120.0	1.65	0.68	0.53	1.46	9.30	0.60	1.62	840.0	class_2
177	14.13	4.10	2.74	24.5	96.0	2.05	0.76	0.56	1.35	9.20	0.61	1.60	560.0	class_2

178 rows × 14 columns

2. **Handwritten Digit Dataset**
    - Features: integers 0-16
    - Classes: 10
    - Total Samples: 1797

Sample Data:



Implement and compare the following ML classifiers for all the 2 datasets and show the classification results (Accuracy, Precision, Recall, F-score, confusion matrix) with and without
parameter tuning:

1. SVM classifier (Linear, Polynomial, Gaussian, & Sigmoid)

Code:

```
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 sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.model_selection import learning_curve

from sklearn.datasets import load_wine

wine = load_wine()

df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df['target'] = wine.target
df['target'] = df['target'].apply(lambda x: wine.target_names[x])

X = df.drop('target', axis=1)
y = df['target']

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=101)

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

classifier = SVC(kernel='linear')

classifier.fit(X_train, y_train)

# Evaluate on test set
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))

classifier = SVC(kernel='poly')

classifier.fit(X_train, y_train)

# Evaluate on test set
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))

classifier = SVC(kernel='rbf')

classifier.fit(X_train, y_train)

# Evaluate on test set
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))

classifier = SVC(kernel='sigmoid')

classifier.fit(X_train, y_train)

# Evaluate on test set
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))

splits = [0.5, 0.4, 0.3, 0.2]
results = []
```

```
for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42, stratify=y
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    svm = SVC(probability=True)

    param_grid = {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    }

    grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)
    y_proba = best_model.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, grid.best_params_.get('kernel'), acc,
prec, rec, f1])

    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        best_model, X_train, y_train, cv=5, scoring="accuracy",
n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_model.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_model.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

results_df = pd.DataFrame(results, columns=["Test Size", "Model Type",
"Accuracy", "Precision", "Recall", "F1"])
display(results_df)

import matplotlib.pyplot as plt

results_df_t = results_df.drop(['Test Size', 'Model Type'], axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size', labels=results_df['Test Size'])
plt.tight_layout()

plt.show()

print("\n=== PCA with Random Forest ===")
```

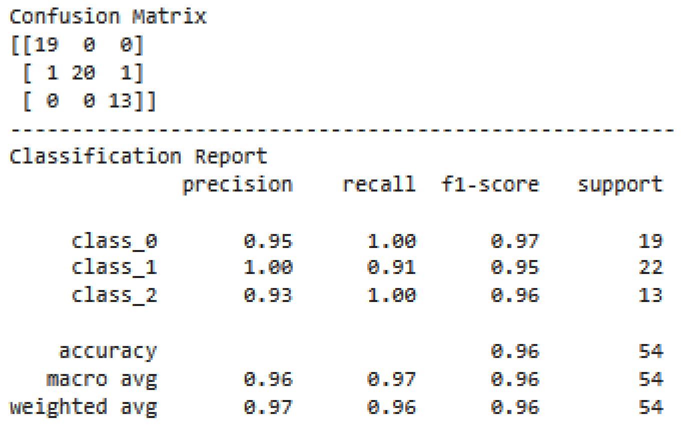
```
pca = PCA(n_components=13)
X_reduced = pca.fit_transform(X)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")

    X_train, X_test, y_train, y_test = train_test_split(
        X_reduced, y, test_size=test_size, random_state=42, stratify=y
    )

    rf = SVC()

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    svm = SVC(probability=True)

    param_grid = {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    }

    grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()

from sklearn.datasets import load_digits

digits = load_digits()

# flatten the images
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))

data.shape
```

```
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.3, shuffle=False
)

classifier = SVC(kernel='linear')

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_pred):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")

classifier = SVC(kernel='poly')

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_pred):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")

classifier = SVC(kernel='rbf')

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_pred):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")

classifier = SVC(kernel='sigmoid')

classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred))

_, axes = plt.subplots(nrows=1, ncols=4, figsize=(10, 3))
for ax, image, prediction in zip(axes, X_test, y_pred):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation="nearest")
    ax.set_title(f"Prediction: {prediction}")

splits = [0.5, 0.4, 0.3, 0.2]
results = []

for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_test, y_train, y_test = train_test_split(
        data, digits.target, test_size=test_size, random_state=42,
shuffle=False
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    svm = SVC(probability=True)
    param_grid = {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    }

    grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)
    y_proba = best_model.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, grid.best_params_.get('kernel'), acc,
prec, rec, f1])

    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        best_model, X_train, y_train, cv=5, scoring="accuracy",
n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_model.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_model.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

results_df = pd.DataFrame(results, columns=["Test Size", "Model Type",
"Accuracy", "Precision", "Recall", "F1"])
display(results_df)

import matplotlib.pyplot as plt

results_df_t = results_df.drop(['Test Size', 'Model Type'], axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size', labels=results_df['Test Size'])
plt.tight_layout()

plt.show()

pca = PCA(n_components=50)
X_reduced = pca.fit_transform(data)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")
    X_train, X_test, y_train, y_test = train_test_split(
        X_reduced, digits.target, test_size=test_size, shuffle=False
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    svm = SVC(probability=True)
    param_grid = {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    }

    grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()

best_accuracy = -1
best_n_components = 0
best_kernel = ""
best_model = None
best_classification_report = ""

for n_components in range(1, 64):
    pca = PCA(n_components=n_components)
    X_reduced = pca.fit_transform(data)

    X_train, X_test, y_train, y_test = train_test_split(
        X_reduced, digits.target, test_size=0.3, shuffle=False
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    svm = SVC(probability=True)

    param_grid = {
        'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    }

    grid = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model_for_this_n = grid.best_estimator_
    best_params = grid.best_params_
    y_pred = best_model_for_this_n.predict(X_test)

    accuracy = grid.best_score_

    if accuracy > best_accuracy:
        best_accuracy = accuracy
        best_n_components = n_components
        best_kernel = best_params['kernel']
        best_model = best_model_for_this_n
        best_classification_report = classification_report(y_test,
y_pred)

print("\n--- Best Results ---")
print(f"Best PCA n_components: {best_n_components}")
print(f"Best Kernel: {best_kernel}")
print(f"Best Accuracy: {best_accuracy}")
print("Best Classification Report:")
print(best_classification_report)
```

Results and Discussion

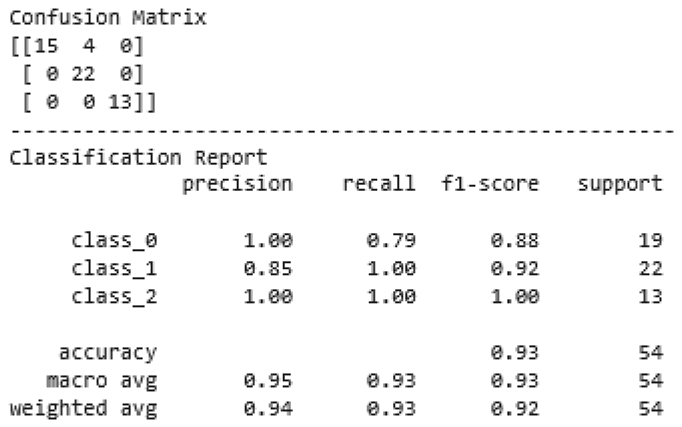
Wine Dataset

Linear SVM:



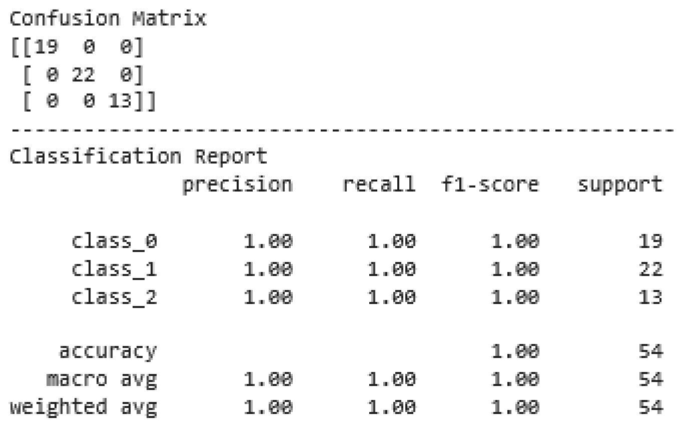
```
Confusion Matrix
[[19  0  0]
 [ 1 20  1]
 [ 0  0 13]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       0.95      1.00      0.97        19
     class_1       1.00      0.91      0.95        22
     class_2       0.93      1.00      0.96        13

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.97      0.96      0.96        54
```

Polynomial SVM



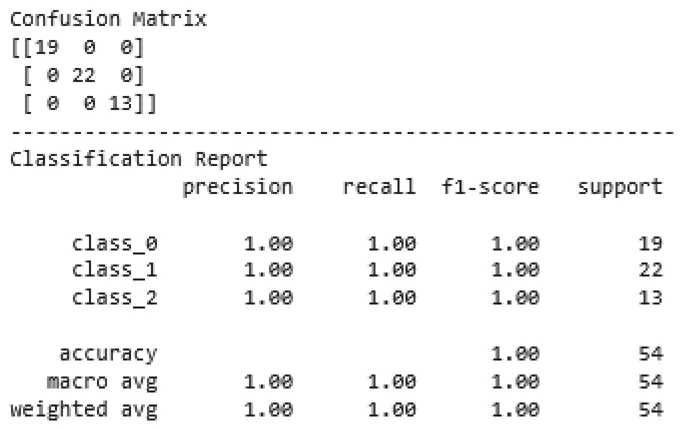
```
Confusion Matrix
[[15  4  0]
 [ 0 22  0]
 [ 0  0 13]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       1.00      0.79      0.88        19
     class_1       0.85      1.00      0.92        22
     class_2       1.00      1.00      1.00        13

    accuracy                           0.93        54
   macro avg       0.95      0.93      0.93        54
weighted avg       0.94      0.93      0.92        54
```

Gaussian SVM



```
Confusion Matrix
[[19  0  0]
 [ 0 22  0]
 [ 0  0 13]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       1.00      1.00      1.00        22
     class_2       1.00      1.00      1.00        13

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

Sigmoid SVM



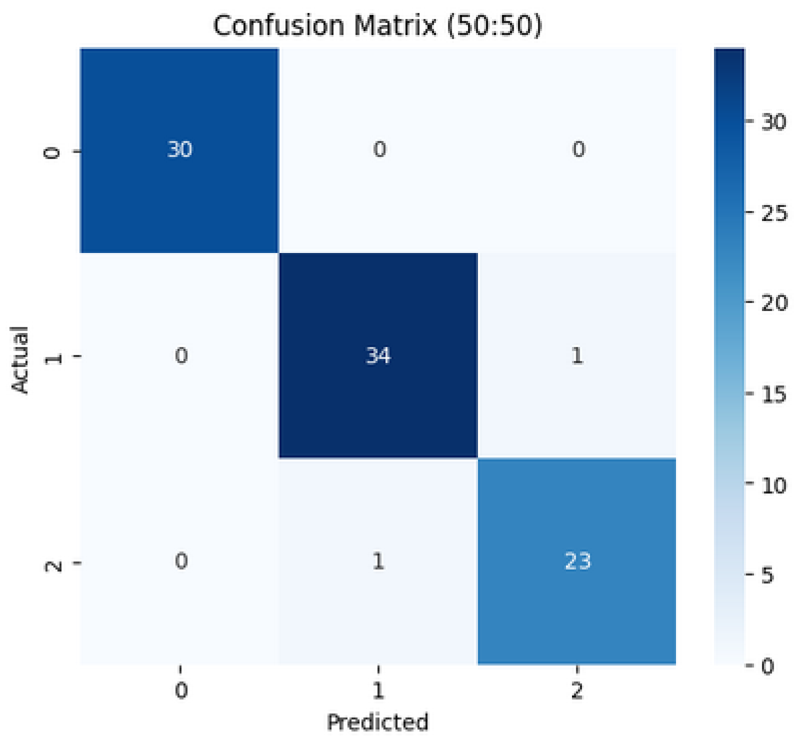
```
Confusion Matrix
[[19  0  0]
 [ 0 22  0]
 [ 0  0 13]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        19
     class_1       1.00      1.00      1.00        22
     class_2       1.00      1.00      1.00        13

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

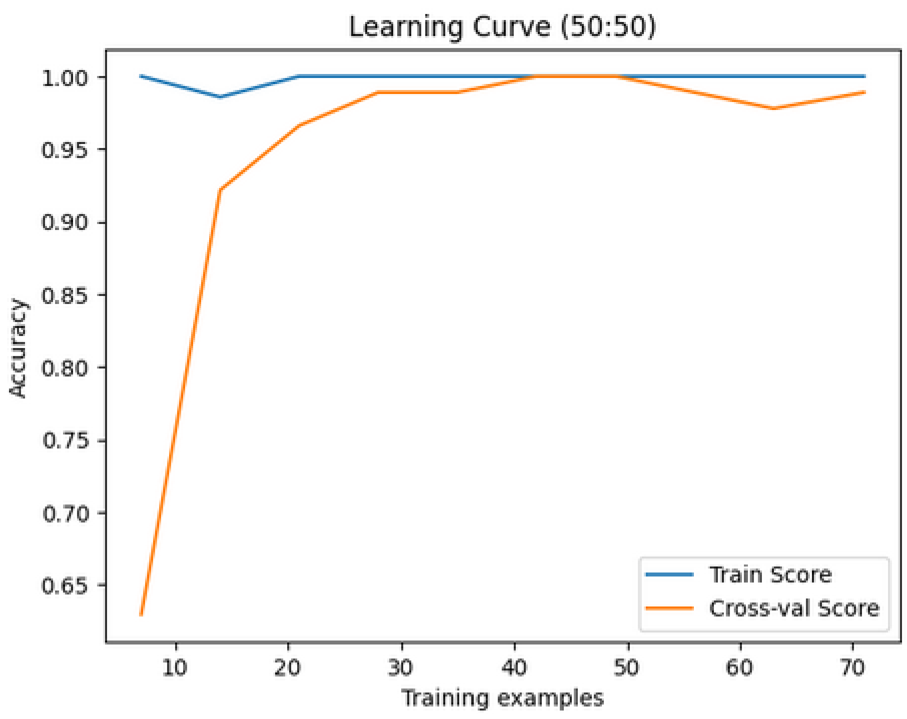
Comparison of different split sizes:

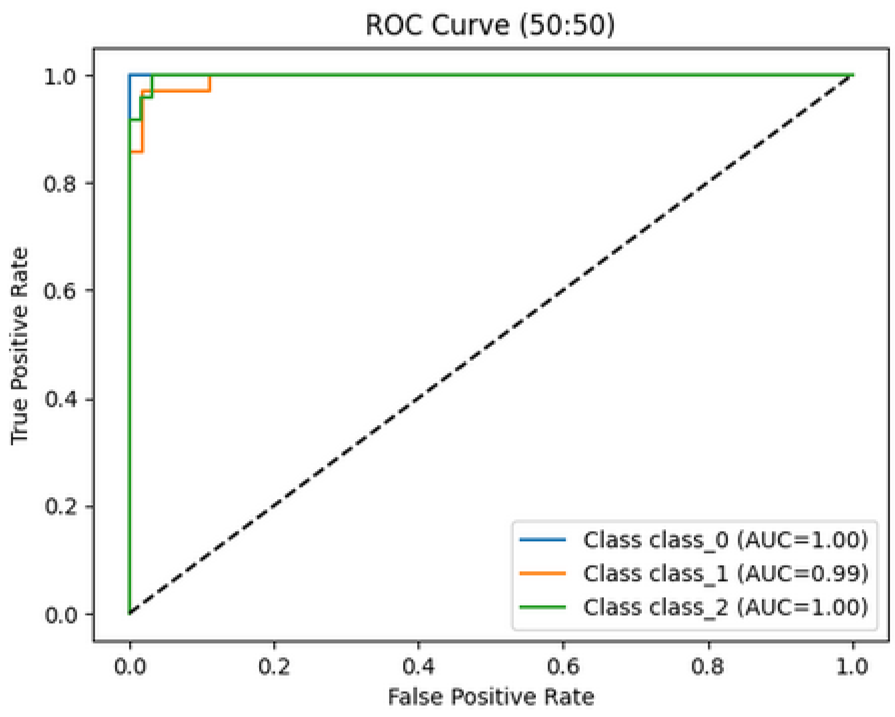
For each test size, the most appropriate kernel has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

Train Test Split (50:50)

```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        30
     class_1       0.97      0.97      0.97        35
     class_2       0.96      0.96      0.96        24

    accuracy                           0.98        89
   macro avg       0.98      0.98      0.98        89
weighted avg       0.98      0.98      0.98        89
```

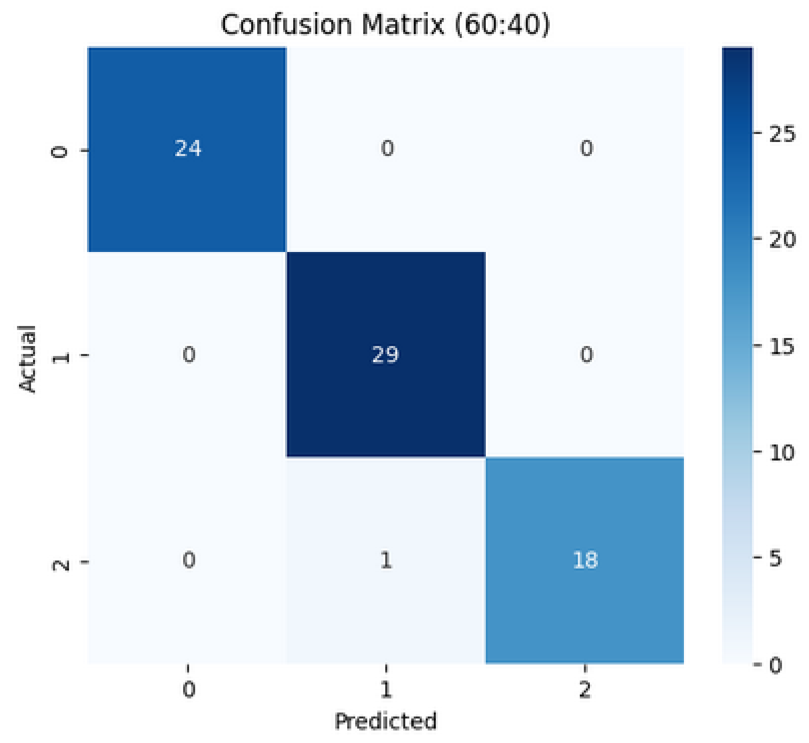


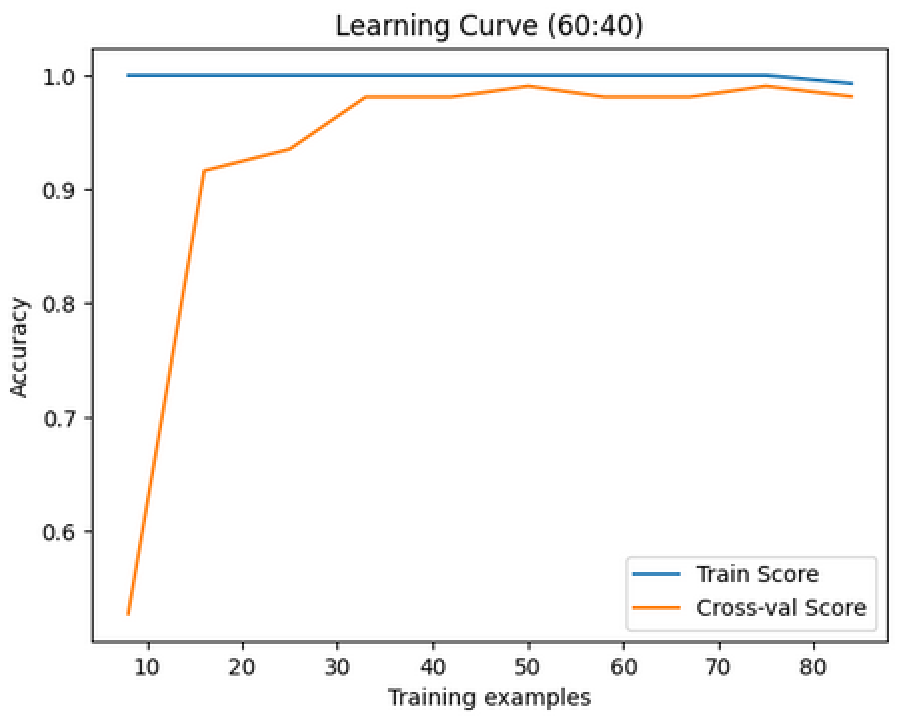


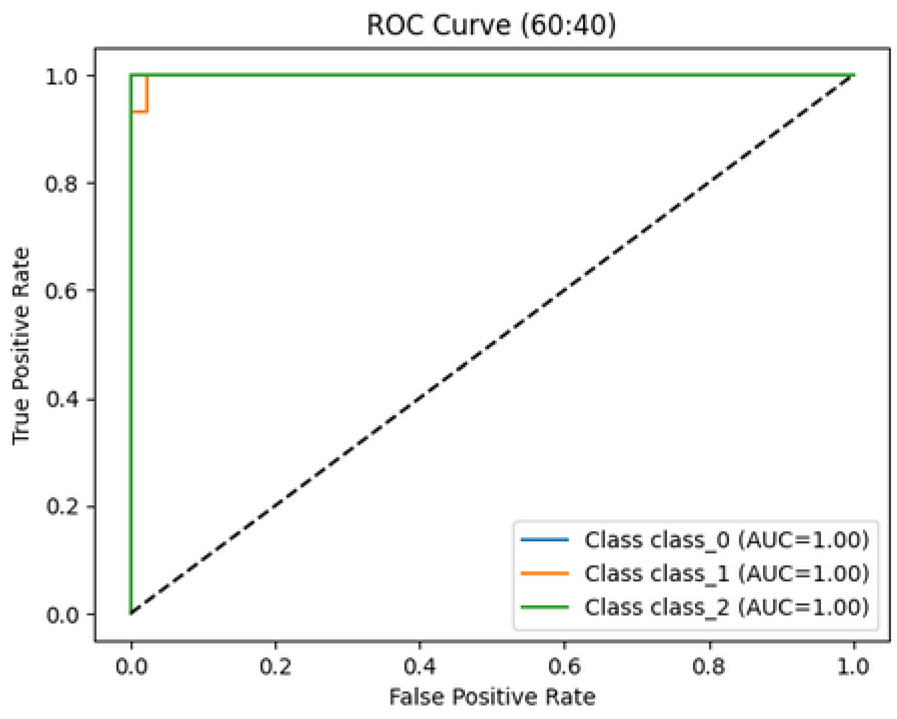


Train Test Split (60:40)

```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        24
     class_1       0.97      1.00      0.98        29
     class_2       1.00      0.95      0.97        19

    accuracy                           0.99        72
   macro avg       0.99      0.98      0.99        72
weighted avg       0.99      0.99      0.99        72
```

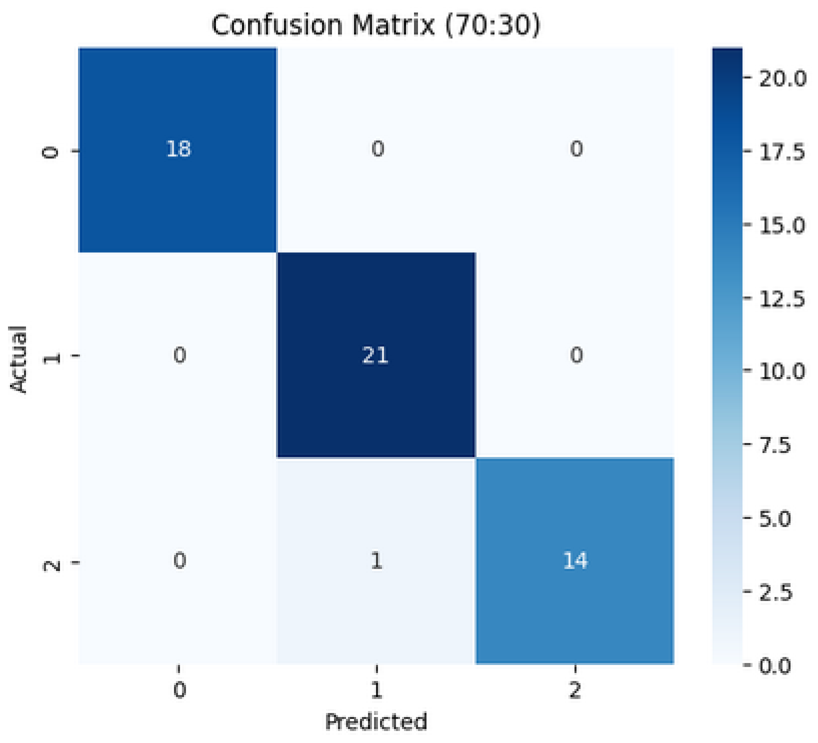


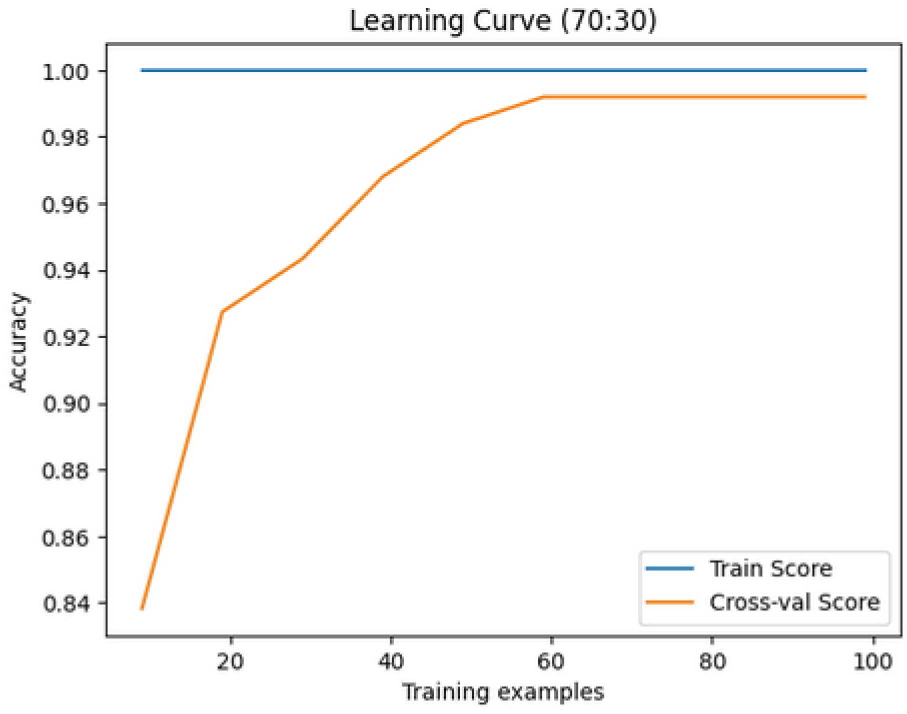


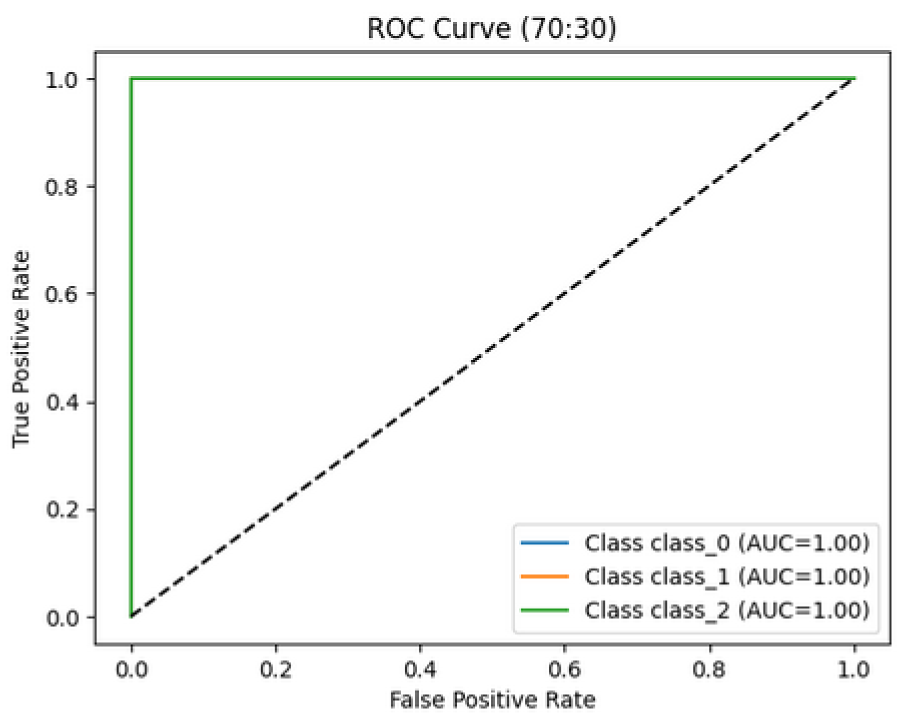


Train Test Split (70:30)

```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        18
     class_1       0.95      1.00      0.98        21
     class_2       1.00      0.93      0.97        15

    accuracy                           0.98        54
   macro avg       0.98      0.98      0.98        54
weighted avg       0.98      0.98      0.98        54
```

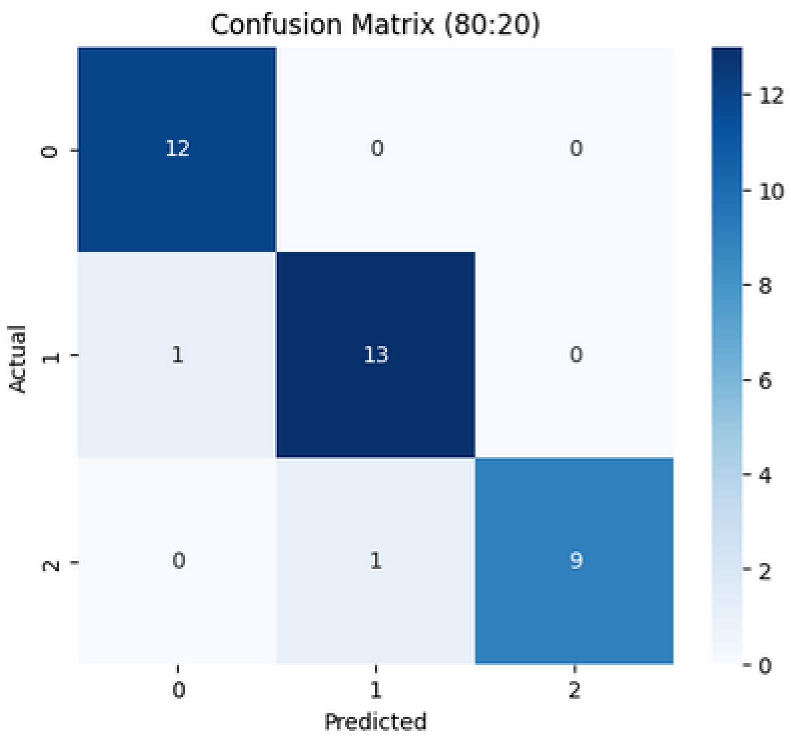

Confusion Matrix (70:30)
Actual
Predicted
18
0
0
0
21
0
0
1
14
0
1
2
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0

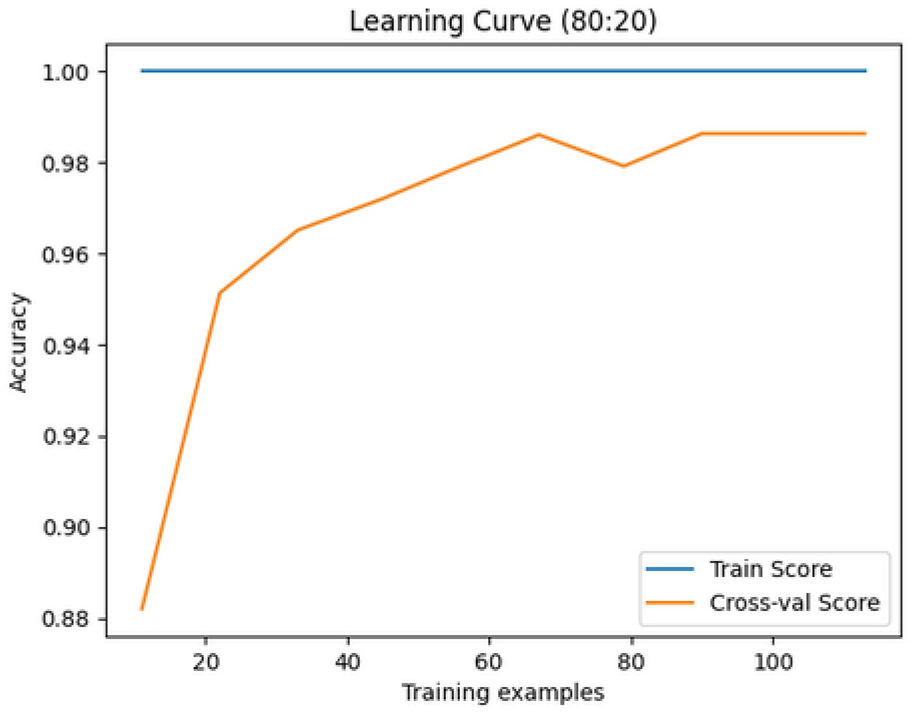

Learning Curve (70:30)
Accuracy
Training examples
Train Score
Cross-val Score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
20
40
60
80
100

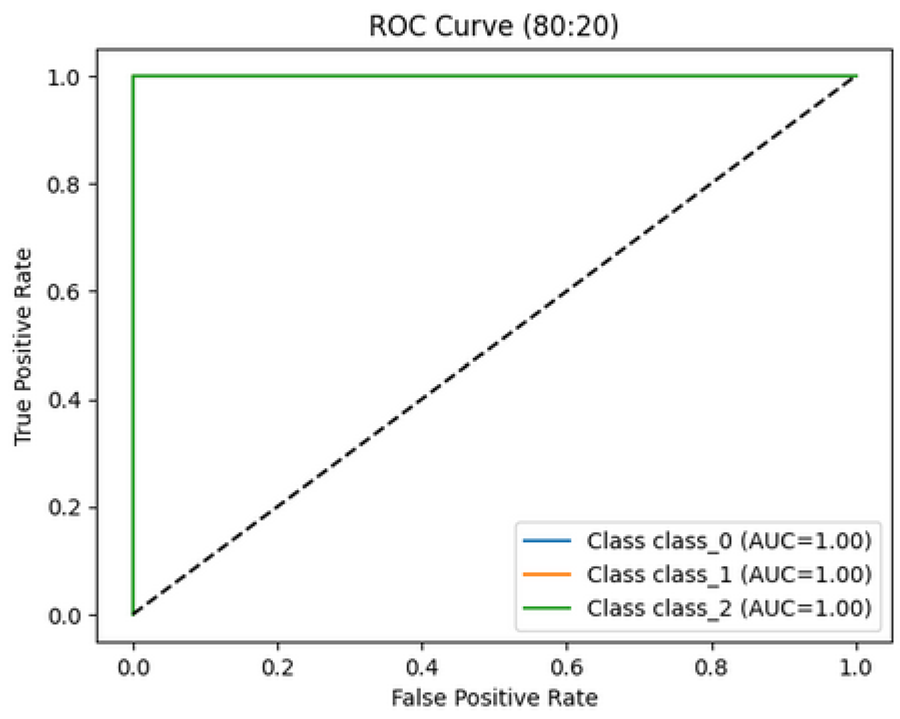


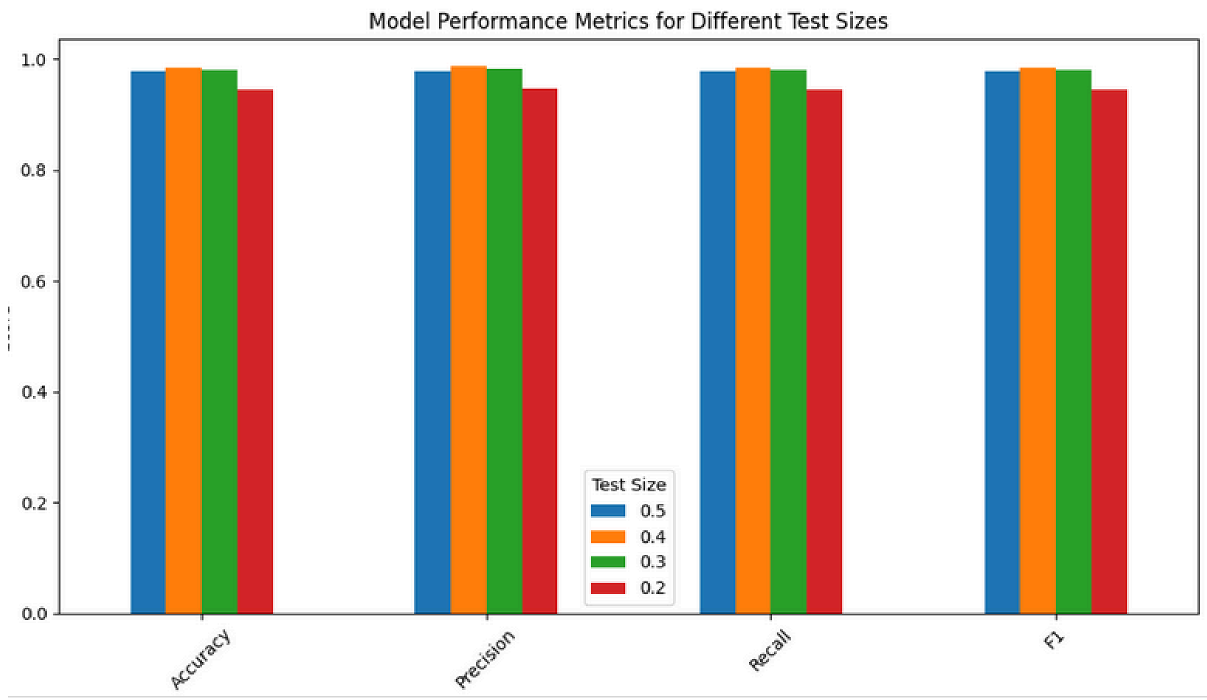
Train Test Split (80:20)

```
Best Parameters: {'kernel': 'linear'}
              precision    recall  f1-score   support

     class_0       0.92      1.00      0.96        12
     class_1       0.93      0.93      0.93        14
     class_2       1.00      0.90      0.95        10

    accuracy                           0.94        36
   macro avg       0.95      0.94      0.95        36
weighted avg       0.95      0.94      0.94        36
```


Confusion Matrix (80:20)
Actual
Predicted
0
1
2
12
0
0
1
13
0
0
1
9
12
10
8
6
4
2
0

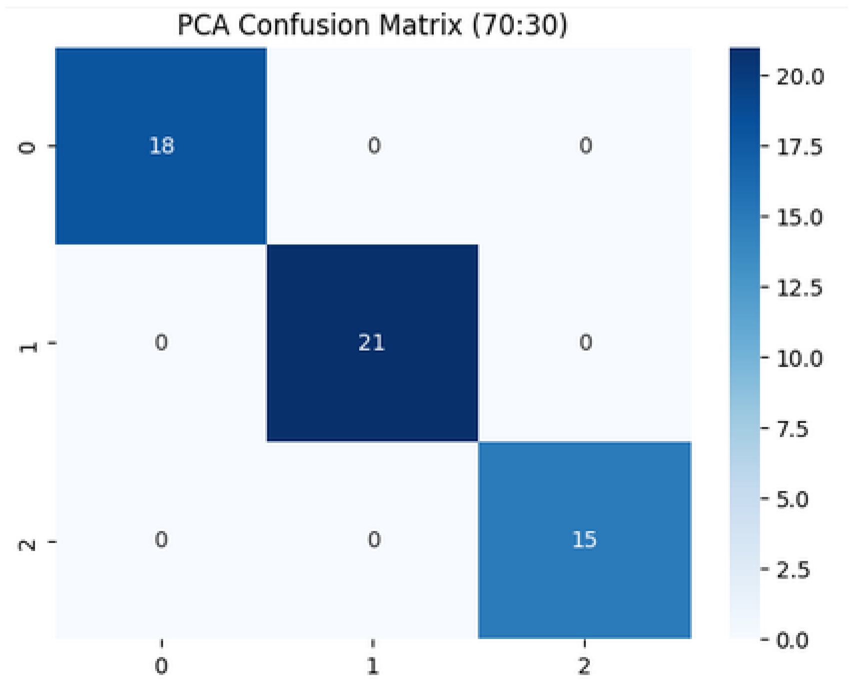

Learning Curve (80:20)
Accuracy
Training examples
1.00
0.98
0.96
0.94
0.92
0.90
0.88
20
40
60
80
100
Train Score
Cross-val Score





Principal Component Analysis (PCA) for feature dimensionality reduction

```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        18
     class_1       1.00      1.00      1.00        21
     class_2       1.00      1.00      1.00        15

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

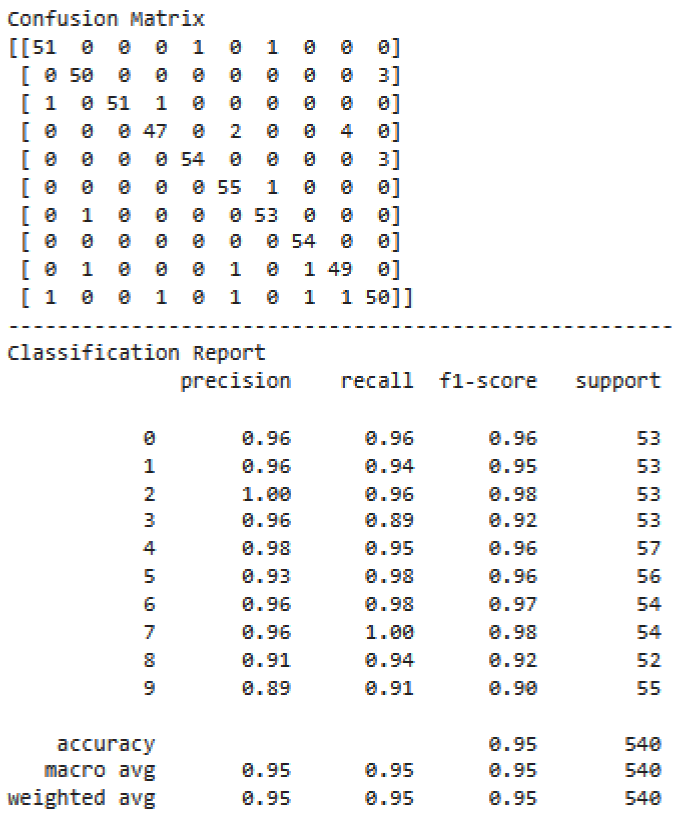


Digits Dataset

Linear SVM:

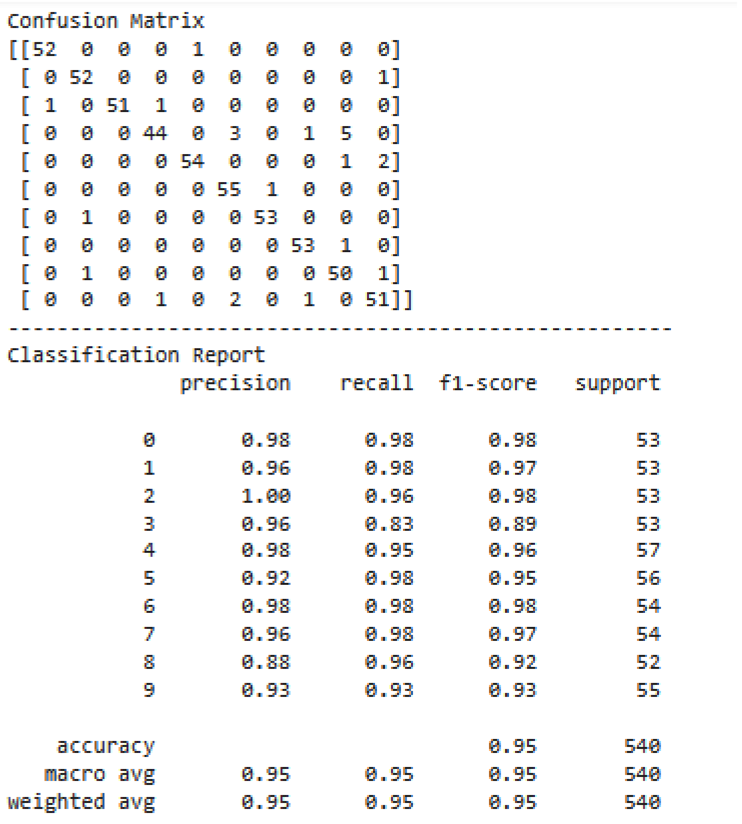
```
Confusion Matrix
[[52  0  0  0  0  0  1  0  0  0]
 [ 0 48  0  0  0  0  0  0  1  4]
 [ 1  0 51  1  0  0  0  0  0  0]
 [ 0  0  0 45  0  1  0  0  7  0]
 [ 1  0  0  0 53  0  0  0  1  2]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  0  0 53  0  0  0]
 [ 0  1  0  0  0  0  0 52  0  1]
 [ 0  2  0  3  1  0  0  1 45  0]
 [ 0  0  0  1  0  2  0  1  1 50]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.98      0.97        53
           1       0.92      0.91      0.91        53
           2       1.00      0.96      0.98        53
           3       0.90      0.85      0.87        53
           4       0.98      0.93      0.95        57
           5       0.95      0.98      0.96        56
           6       0.96      0.98      0.97        54
           7       0.96      0.96      0.96        54
           8       0.82      0.87      0.84        52
           9       0.88      0.91      0.89        55

    accuracy                           0.93       540
   macro avg       0.93      0.93      0.93       540
weighted avg       0.93      0.93      0.93       540
```

Polynomial SVM



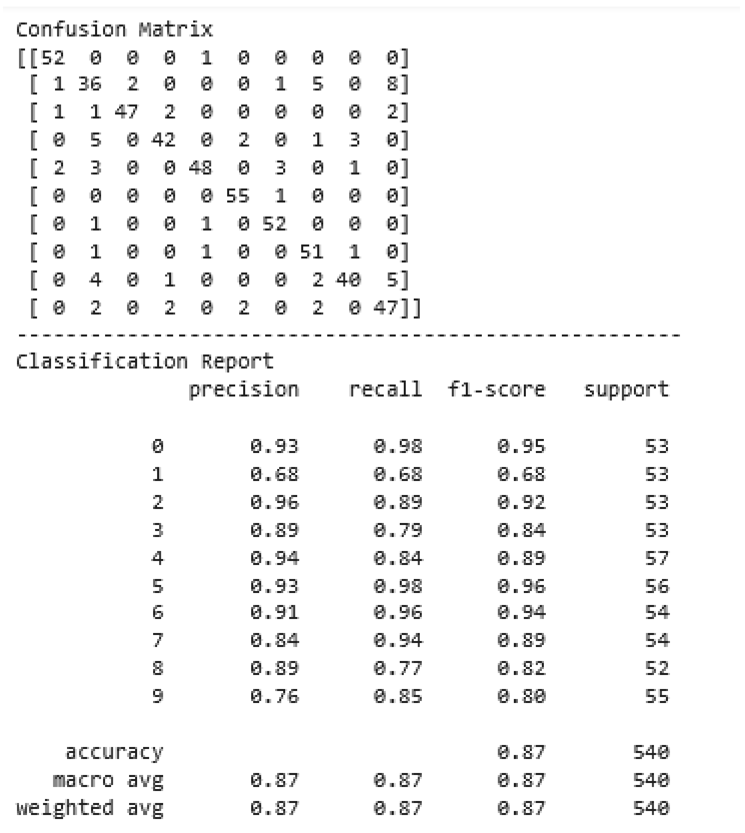
```
Confusion Matrix
[[51  0  0  0  1  0  1  0  0  0]
 [ 0 50  0  0  0  0  0  0  0  3]
 [ 1  0 51  1  0  0  0  0  0  0]
 [ 0  0  0 47  0  2  0  0  4  0]
 [ 0  0  0  0 54  0  0  0  0  3]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 54  0  0]
 [ 0  1  0  0  0  1  0  1 49  0]
 [ 1  0  0  1  0  1  0  1  1 50]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.96      0.96      0.96        53
           1       0.96      0.94      0.95        53
           2       1.00      0.96      0.98        53
           3       0.96      0.89      0.92        53
           4       0.98      0.95      0.96        57
           5       0.93      0.98      0.96        56
           6       0.96      0.98      0.97        54
           7       0.96      1.00      0.98        54
           8       0.91      0.94      0.92        52
           9       0.89      0.91      0.90        55

    accuracy                           0.95       540
   macro avg       0.95      0.95      0.95       540
weighted avg       0.95      0.95      0.95       540
```

Gaussian SVM



```
Confusion Matrix
[[52  0  0  0  1  0  0  0  0  0]
 [ 0 52  0  0  0  0  0  0  0  1]
 [ 1  0 51  1  0  0  0  0  0  0]
 [ 0  0  0 44  0  3  0  1  5  0]
 [ 0  0  0  0 54  0  0  0  1  2]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 53  1  0]
 [ 0  1  0  0  0  0  0  0 50  1]
 [ 0  0  0  1  0  2  0  1  0 51]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        53
           1       0.96      0.98      0.97        53
           2       1.00      0.96      0.98        53
           3       0.96      0.83      0.89        53
           4       0.98      0.95      0.96        57
           5       0.92      0.98      0.95        56
           6       0.98      0.98      0.98        54
           7       0.96      0.98      0.97        54
           8       0.88      0.96      0.92        52
           9       0.93      0.93      0.93        55

    accuracy                           0.95       540
   macro avg       0.95      0.95      0.95       540
weighted avg       0.95      0.95      0.95       540
```

Sigmoid SVM

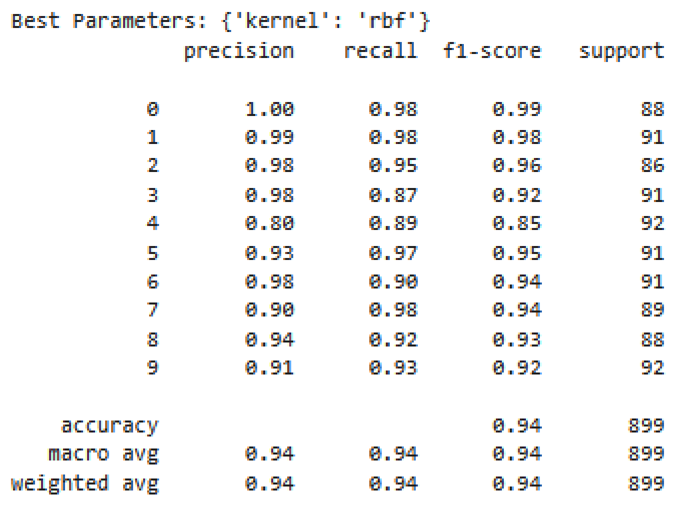


```
Confusion Matrix
[[52  0  0  0  1  0  0  0  0  0]
 [ 1 36  2  0  0  0  1  5  0  8]
 [ 1  1 47  2  0  0  0  0  0  2]
 [ 0  5  0 42  0  2  0  1  3  0]
 [ 2  3  0  0 48  0  3  0  1  0]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  1  0 52  0  0  0]
 [ 0  1  0  0  1  0  0 51  1  0]
 [ 0  4  0  1  0  0  0  2 40  5]
 [ 0  2  0  2  0  2  0  2  0 47]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.93      0.98      0.95        53
           1       0.68      0.68      0.68        53
           2       0.96      0.89      0.92        53
           3       0.89      0.79      0.84        53
           4       0.94      0.84      0.89        57
           5       0.93      0.98      0.96        56
           6       0.91      0.96      0.94        54
           7       0.84      0.94      0.89        54
           8       0.89      0.77      0.82        52
           9       0.76      0.85      0.80        55

    accuracy                           0.87       540
   macro avg       0.87      0.87      0.87       540
weighted avg       0.87      0.87      0.87       540
```

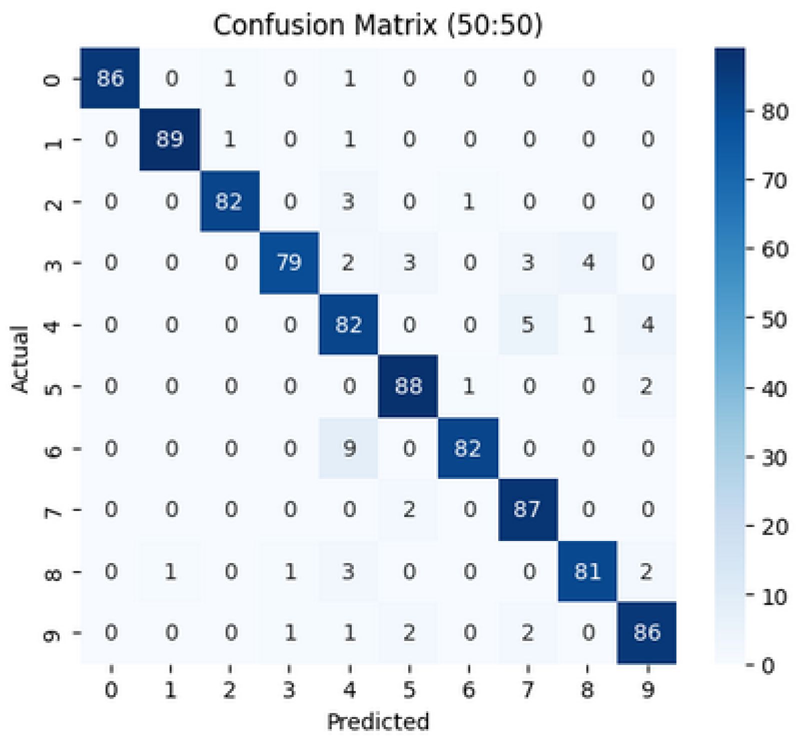
Comparison of different split sizes:

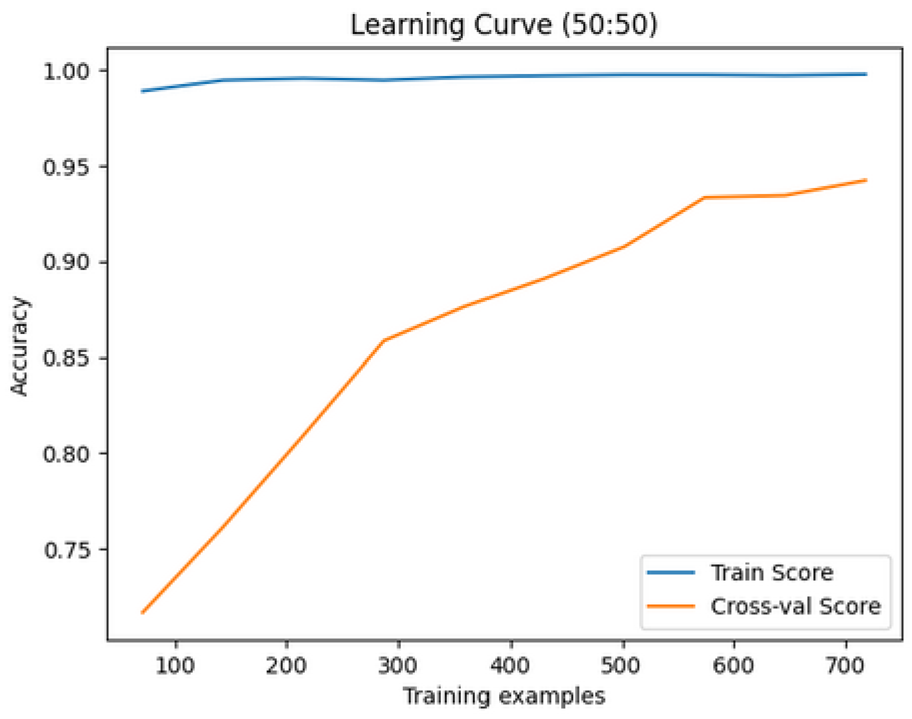
For each test size, the most appropriate kernel has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

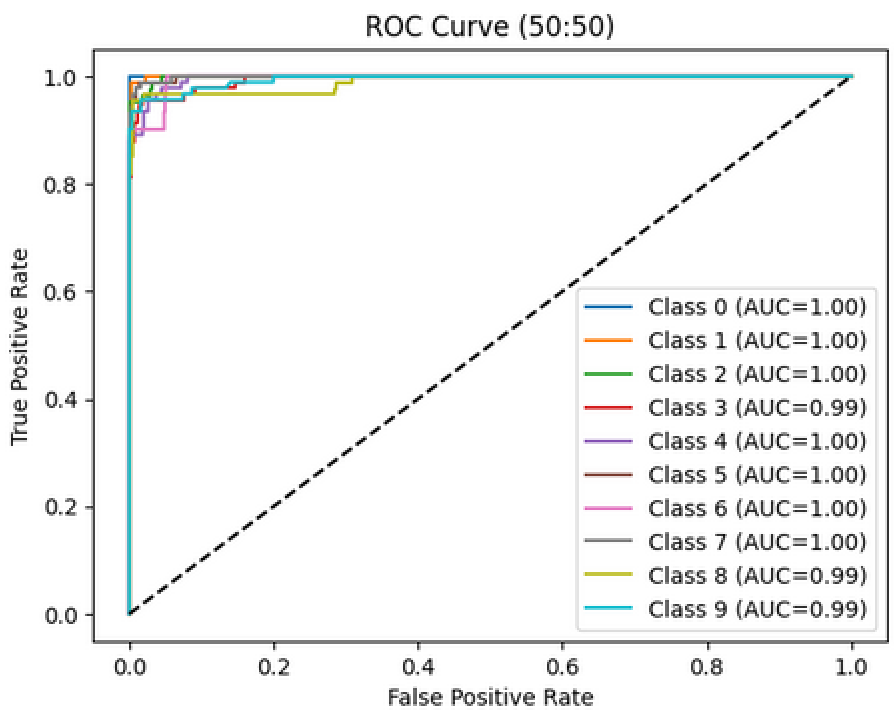
Train Test Split (50:50)



```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        88
           1       0.99      0.98      0.98        91
           2       0.98      0.95      0.96        86
           3       0.98      0.87      0.92        91
           4       0.80      0.89      0.85        92
           5       0.93      0.97      0.95        91
           6       0.98      0.90      0.94        91
           7       0.90      0.98      0.94        89
           8       0.94      0.92      0.93        88
           9       0.91      0.93      0.92        92

    accuracy                           0.94       899
   macro avg       0.94      0.94      0.94       899
weighted avg       0.94      0.94      0.94       899
```

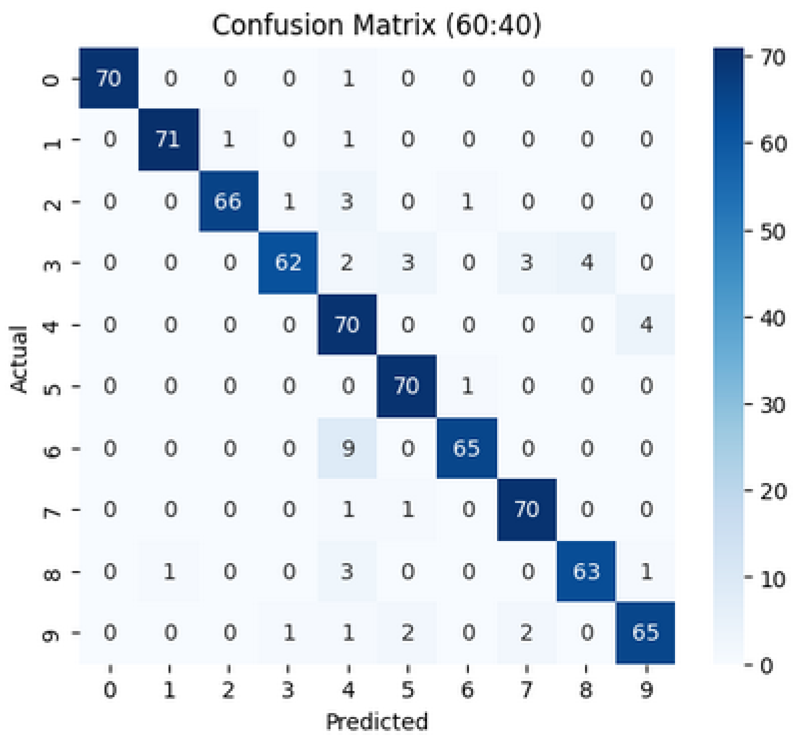

Confusion Matrix (50:50)
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
86 0 1 0 1 0 0 0 0 0
0 89 1 0 1 0 0 0 0 0
0 0 82 0 3 0 1 0 0 0
0 0 0 79 2 3 0 3 4 0
0 0 0 0 82 0 0 5 1 4
0 0 0 0 0 88 1 0 0 2
0 0 0 0 9 0 82 0 0 0
0 0 0 0 0 2 0 87 0 0
0 1 0 1 3 0 0 0 81 2
0 0 0 1 1 2 0 2 0 86
0 10 20 30 40 50 60 70 80


Learning Curve (50:50)
Accuracy
Training examples
1.00
0.95
0.90
0.85
0.80
0.75
100 200 300 400 500 600 700
Train Score
Cross-val Score

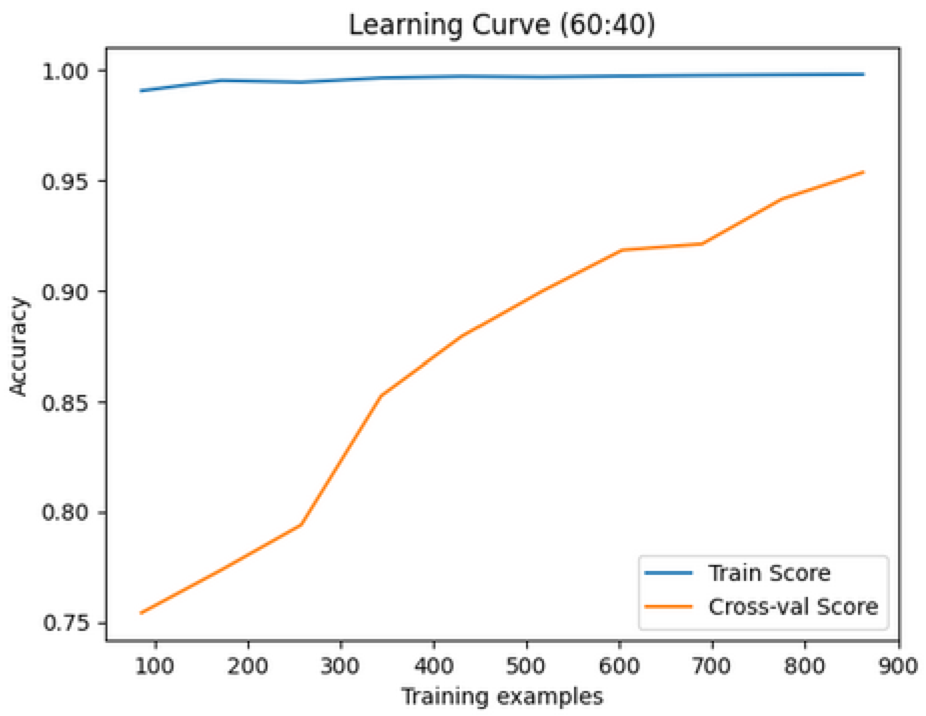


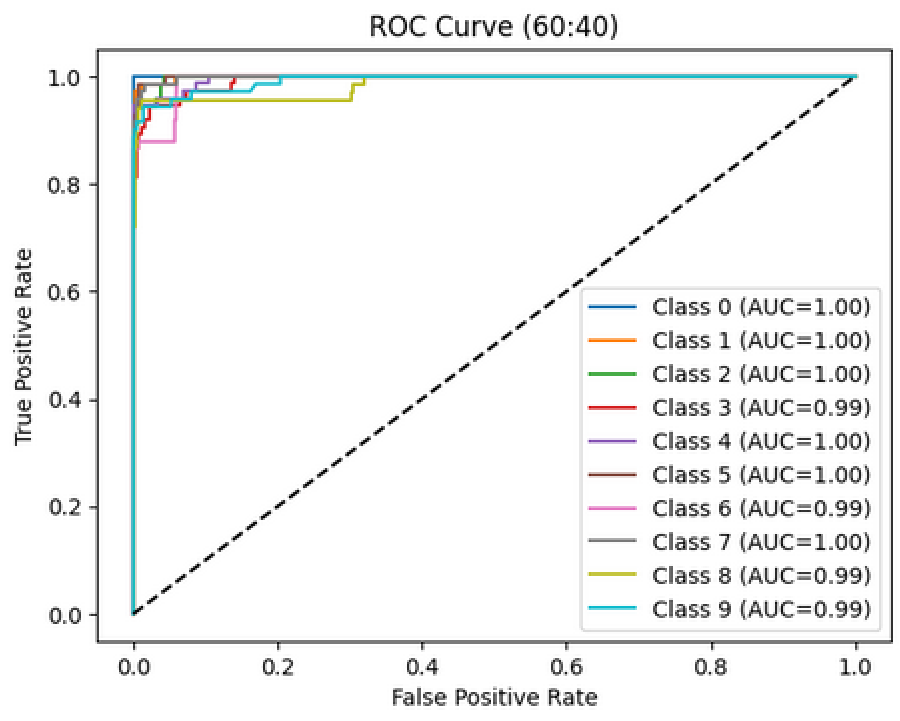
Train Test Split (60:40)

```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

           0       1.00      0.99      0.99        71
           1       0.99      0.97      0.98        73
           2       0.99      0.93      0.96        71
           3       0.97      0.84      0.90        74
           4       0.77      0.95      0.85        74
           5       0.92      0.99      0.95        71
           6       0.97      0.88      0.92        74
           7       0.93      0.97      0.95        72
           8       0.94      0.93      0.93        68
           9       0.93      0.92      0.92        71

    accuracy                           0.93       719
   macro avg       0.94      0.94      0.94       719
weighted avg       0.94      0.93      0.94       719
```



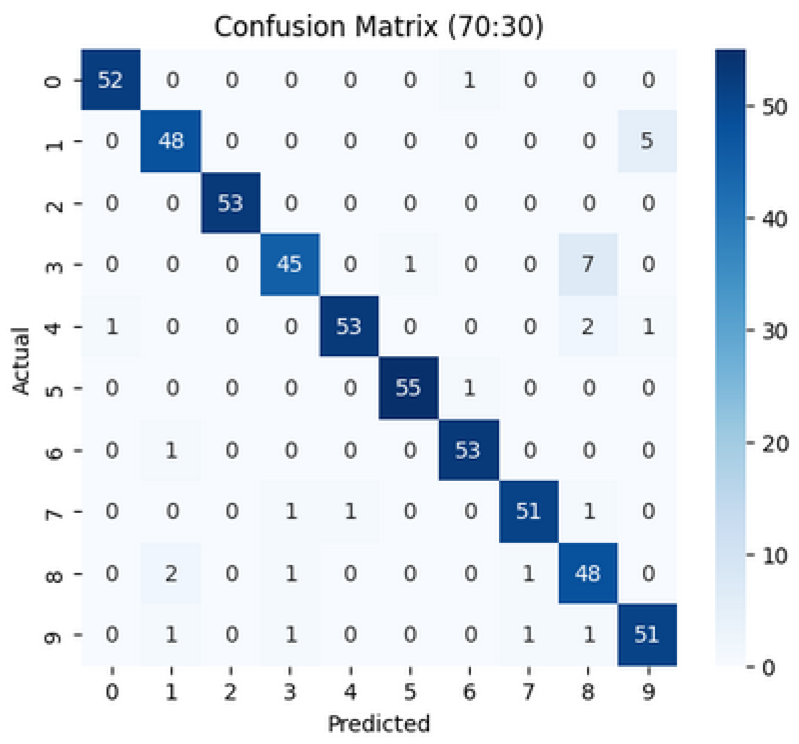
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	70	0	0	0	1	0	0	0	0	0
1	0	71	1	0	1	0	0	0	0	0
2	0	0	66	1	3	0	1	0	0	0
3	0	0	0	62	2	3	0	3	4	0
4	0	0	0	0	70	0	0	0	0	4
5	0	0	0	0	0	70	1	0	0	0
6	0	0	0	0	9	0	65	0	0	0
7	0	0	0	0	1	1	0	70	0	0
8	0	1	0	0	3	0	0	0	63	1
9	0	0	0	1	1	2	0	2	0	65

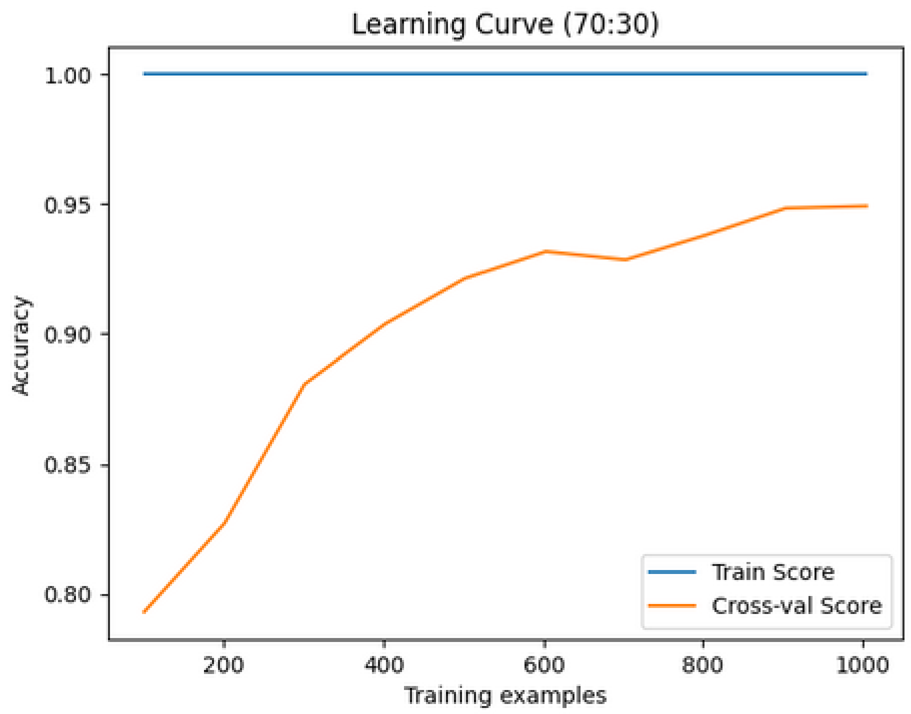


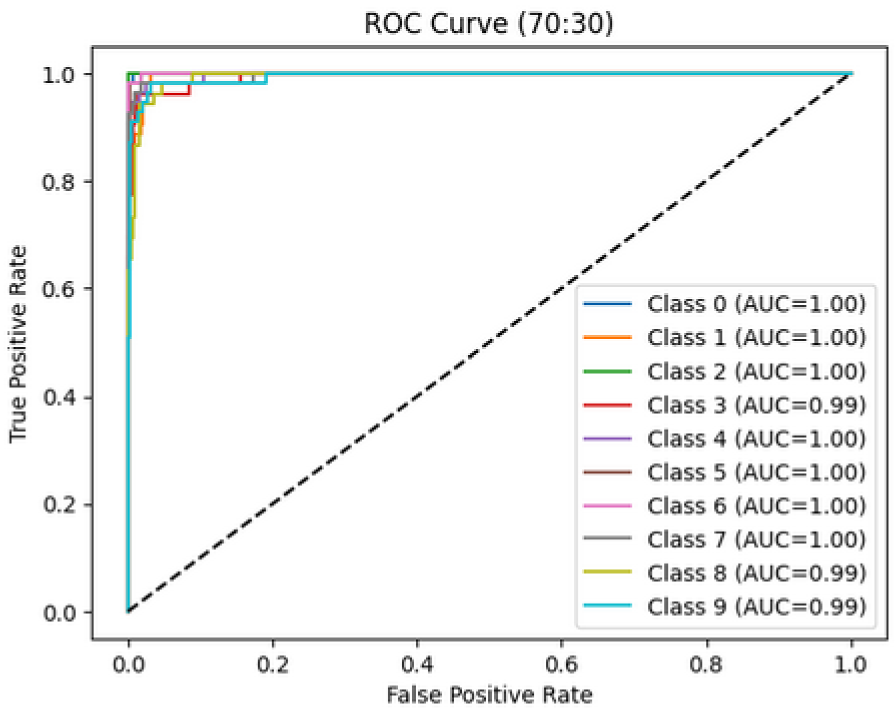


Train Test Split (70:30)

```
Best Parameters: {'kernel': 'linear'}
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        53
           1       0.92      0.91      0.91        53
           2       1.00      1.00      1.00        53
           3       0.94      0.85      0.89        53
           4       0.98      0.93      0.95        57
           5       0.98      0.98      0.98        56
           6       0.96      0.98      0.97        54
           7       0.96      0.94      0.95        54
           8       0.81      0.92      0.86        52
           9       0.89      0.93      0.91        55

    accuracy                           0.94       540
   macro avg       0.94      0.94      0.94       540
weighted avg       0.94      0.94      0.94       540
```

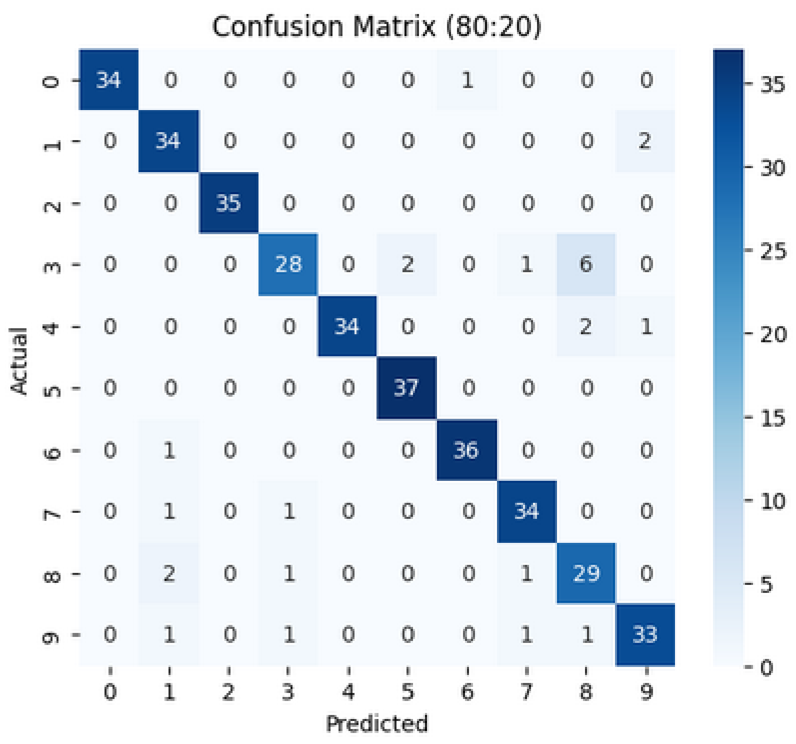

Confusion Matrix (70:30)
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
0: 52 0 0 0 0 0 1 0 0 0
1: 0 48 0 0 0 0 0 0 0 5
2: 0 0 53 0 0 0 0 0 0 0
3: 0 0 0 45 0 1 0 0 7 0
4: 1 0 0 0 53 0 0 0 2 1
5: 0 0 0 0 0 55 1 0 0 0
6: 0 1 0 0 0 0 53 0 0 0
7: 0 0 0 1 1 0 0 51 1 0
8: 0 2 0 1 0 0 0 1 48 0
9: 0 1 0 1 0 0 0 1 1 51
0
10
20
30
40
50

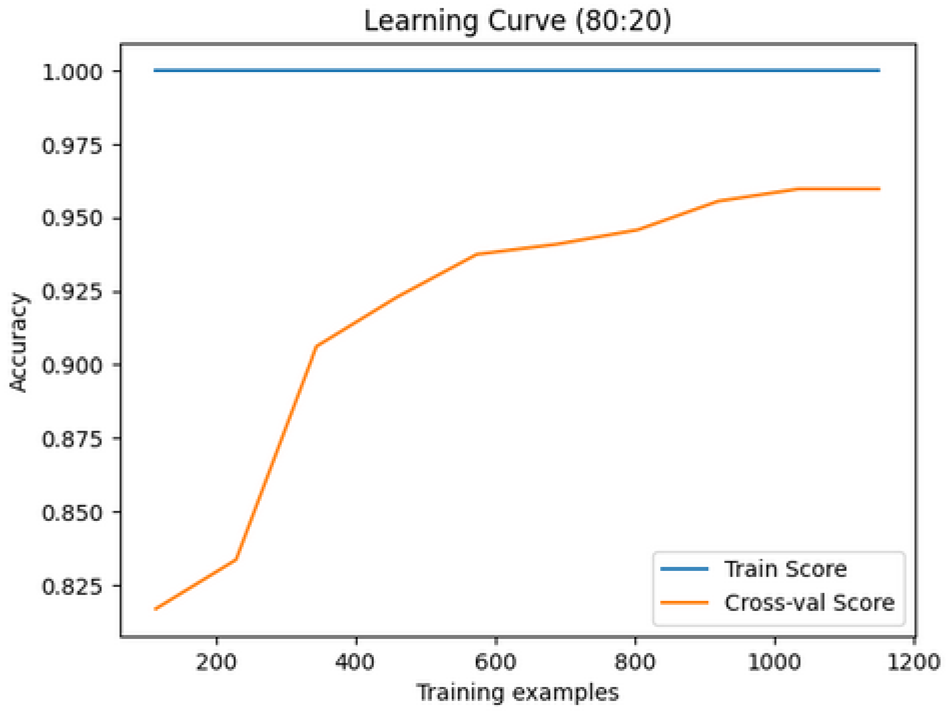

Learning Curve (70:30)
Accuracy
Training examples
1.00
0.95
0.90
0.85
0.80
200
400
600
800
1000
Train Score
Cross-val Score

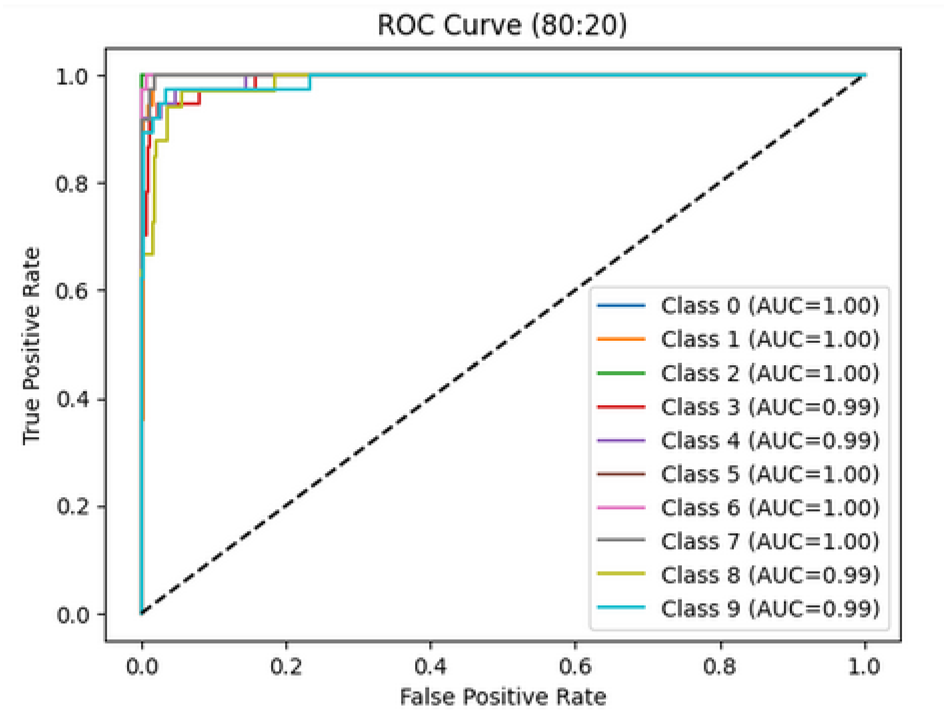


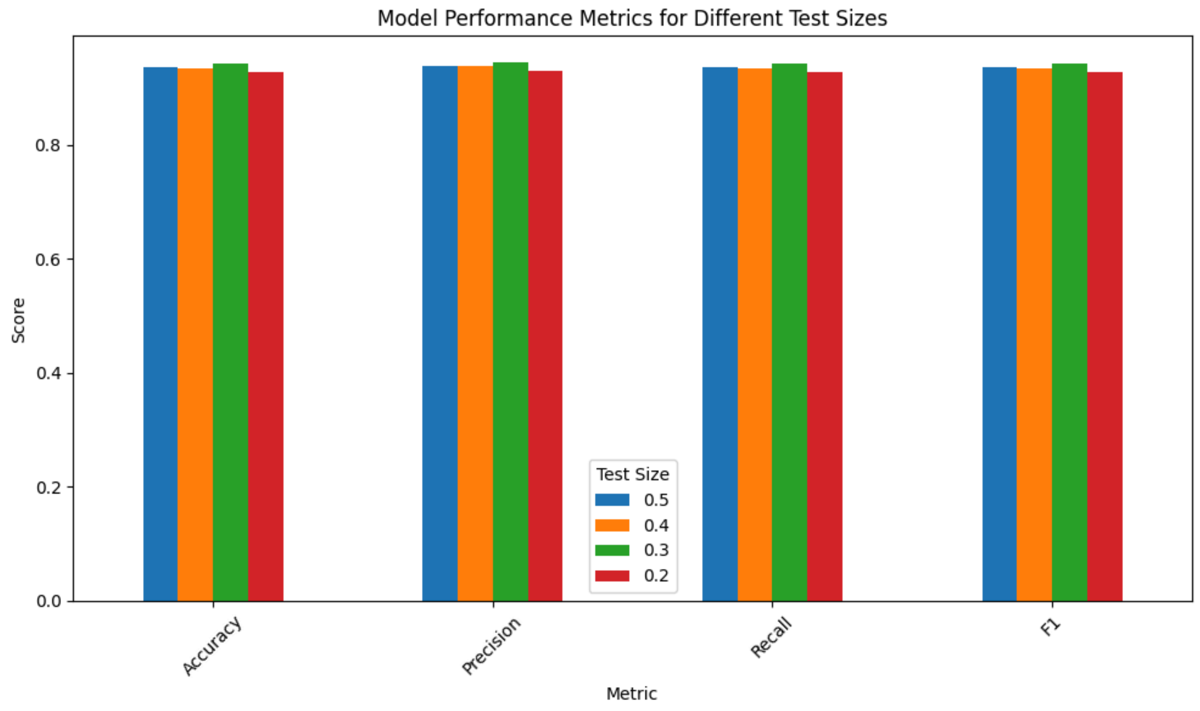
Train Test Split (80:20)

```
Best Parameters: {'kernel': 'linear'}
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        35
           1       0.87      0.94      0.91        36
           2       1.00      1.00      1.00        35
           3       0.90      0.76      0.82        37
           4       1.00      0.92      0.96        37
           5       0.95      1.00      0.97        37
           6       0.97      0.97      0.97        37
           7       0.92      0.94      0.93        36
           8       0.76      0.88      0.82        33
           9       0.92      0.89      0.90        37

    accuracy                           0.93       360
   macro avg       0.93      0.93      0.93       360
weighted avg       0.93      0.93      0.93       360
```

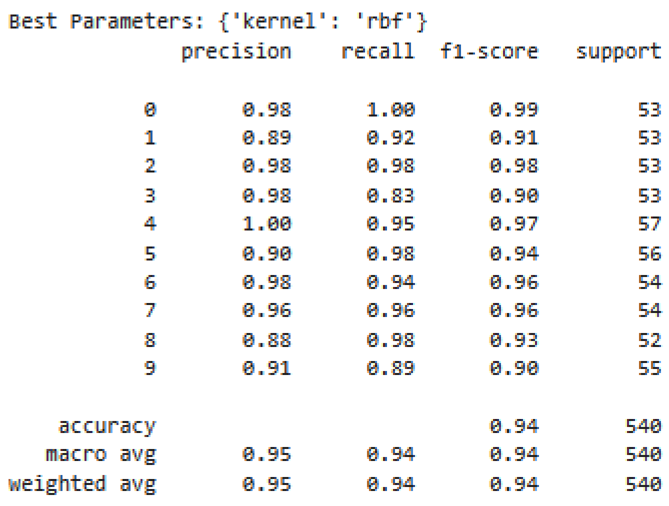

Confusion Matrix (80:20)
Actual
Predicted
0 1 2 3 4 5 6 7 8 9
34 0 0 0 0 0 1 0 0 0
0 34 0 0 0 0 0 0 0 2
0 0 35 0 0 0 0 0 0 0
0 0 0 28 0 2 0 1 6 0
0 0 0 0 34 0 0 0 2 1
0 0 0 0 0 37 0 0 0 0
0 1 0 0 0 0 36 0 0 0
0 1 0 1 0 0 0 34 0 0
0 2 0 1 0 0 0 1 29 0
0 1 0 1 0 0 0 1 1 33
35
30
25
20
15
10
5
0


Learning Curve (80:20)
Accuracy
1.000
0.975
0.950
0.925
0.900
0.875
0.850
0.825
200
400
600
800
1000
1200
Training examples
Train Score
Cross-val Score

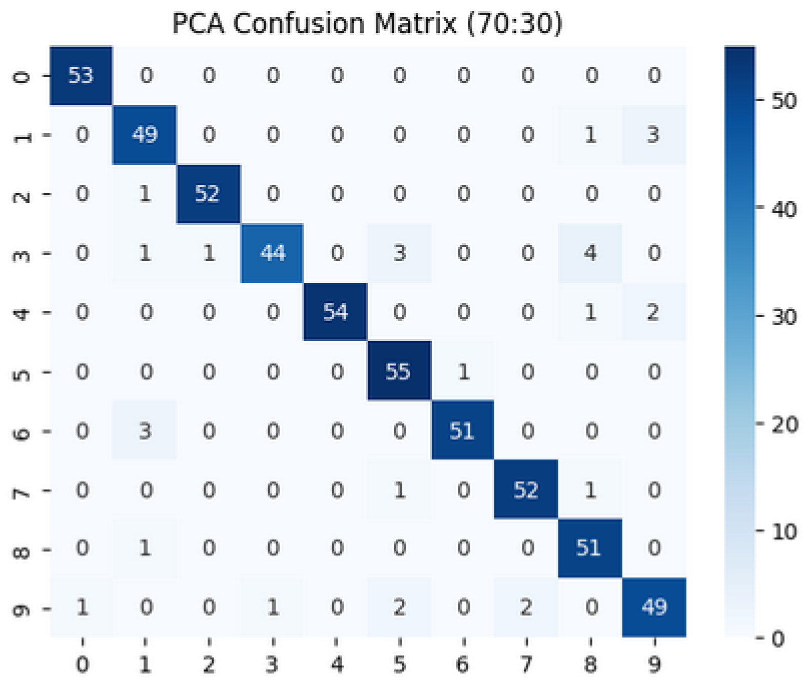




<u>Principal Component Analysis (PCA) for feature dimensionality reduction</u>



```
Best Parameters: {'kernel': 'rbf'}
              precision    recall  f1-score   support

           0       0.98      1.00      0.99        53
           1       0.89      0.92      0.91        53
           2       0.98      0.98      0.98        53
           3       0.98      0.83      0.90        53
           4       1.00      0.95      0.97        57
           5       0.90      0.98      0.94        56
           6       0.98      0.94      0.96        54
           7       0.96      0.96      0.96        54
           8       0.88      0.98      0.93        52
           9       0.91      0.89      0.90        55

    accuracy                           0.94       540
   macro avg       0.95      0.94      0.94       540
weighted avg       0.95      0.94      0.94       540
```



2. MLP classifier (Momentum term, Epoch size and learning rate)

Code:

```
import pandas as pd
import numpy as np
import optuna
import warnings
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.model_selection import learning_curve

from sklearn.datasets import load_wine

wine = load_wine()

warnings.filterwarnings("ignore", category=UserWarning,
module="optuna")

X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=0.3, stratify=y, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, stratify=y_temp, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)

def objective(trial):

    hidden_layer_sizes =
trial.suggest_categorical("hidden_layer_sizes", [(100,), (200,),
(300,), (100,100), (200,100)])
    activation = trial.suggest_categorical("activation", ["relu",
"tanh", "logistic"])
    solver = trial.suggest_categorical("solver", ["adam", "sgd",
"lbfgs"])
    alpha = alpha = trial.suggest_float("alpha", 1e-5, 1e-2, log=True)
    learning_rate = trial.suggest_categorical("learning_rate",
["constant", "adaptive", "invscaling"])

    clf = MLPClassifier(
        hidden_layer_sizes=hidden_layer_sizes,
        activation=activation,
        solver=solver,
        alpha=alpha,
        learning_rate=learning_rate,
        max_iter=1000,
        random_state=42
    )

    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_val)

    return accuracy_score(y_val, y_pred)


# Run Optuna search
```

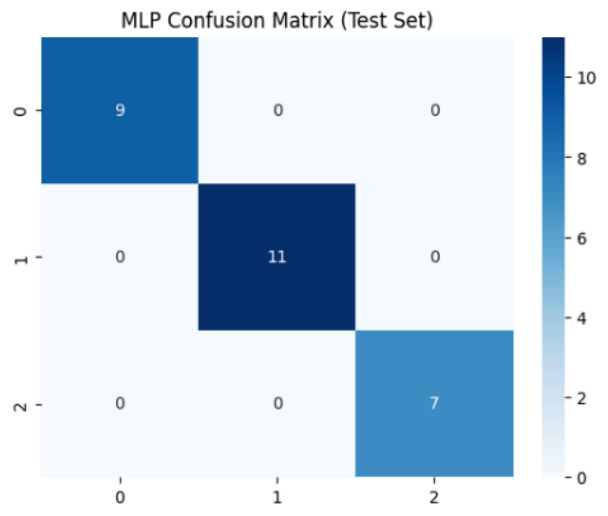
```
study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)

print("\nBest Trial:")
print(study.best_trial.params)

# Train best model
best_params = study.best_trial.params
best_clf = MLPClassifier(**best_params, max_iter=1000, random_state=42)
best_clf.fit(X_train, y_train)

y_pred = best_clf.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-----------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))

sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
plt.title("MLP Confusion Matrix (Test Set)")
plt.show()

splits = [0.5, 0.4, 0.3, 0.2]
results = []
warnings.filterwarnings("ignore", category=UserWarning,
module="optuna")

for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_temp, y_train, y_temp = train_test_split(X, y,
test_size=test_size, stratify=y, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, stratify=y_temp, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    def objective(trial):

        hidden_layer_sizes =
trial.suggest_categorical("hidden_layer_sizes", [(100,), (200,),
(300,), (100,100), (200,100)])
```

```
        activation = trial.suggest_categorical("activation", ["relu",
"tanh", "logistic"])
        solver = trial.suggest_categorical("solver", ["adam", "sgd",
"lbfgs"])
        alpha = alpha = trial.suggest_float("alpha", 1e-5, 1e-2,
log=True)
        learning_rate = trial.suggest_categorical("learning_rate",
["constant", "adaptive", "invscaling"])

        clf = MLPClassifier(
            hidden_layer_sizes=hidden_layer_sizes,
            activation=activation,
            solver=solver,
            alpha=alpha,
            learning_rate=learning_rate,
            max_iter=1000,
            random_state=42
        )

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_val)

        return accuracy_score(y_val, y_pred)

    # Run Optuna search
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)

    print("\nBest Trial:")
    print(study.best_trial.params)

    # Train best model
    best_params = study.best_trial.params
    best_clf = MLPClassifier(**best_params, max_iter=1000,
random_state=42)
    best_clf.fit(X_train, y_train)

    y_pred = best_clf.predict(X_test)
    y_proba = best_clf.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, acc, prec, rec, f1])

    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        best_clf, X_train, y_train, cv=5, scoring="accuracy",
n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_clf.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_clf.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

results_df = pd.DataFrame(results, columns=["Test Size", "Accuracy",
"Precision", "Recall", "F1"])
display(results_df)

import matplotlib.pyplot as plt

results_df_t = results_df.drop(['Test Size'], axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size', labels=results_df['Test Size'])
plt.tight_layout()

plt.show()

print("\n=== PCA with Random Forest ===")

pca = PCA(n_components=5)
X_reduced = pca.fit_transform(X)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")

    X_train, X_temp, y_train, y_temp = train_test_split(X_reduced, y,
test_size=test_size, stratify=y, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, stratify=y_temp, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    def objective(trial):

        hidden_layer_sizes =
trial.suggest_categorical("hidden_layer_sizes", [(100,), (200,),
(300,), (100,100), (200,100)])
        activation = trial.suggest_categorical("activation", ["relu",
"tanh", "logistic"])
        solver = trial.suggest_categorical("solver", ["adam", "sgd",
"lbfgs"])
        alpha = alpha = trial.suggest_float("alpha", 1e-5, 1e-2,
log=True)
        learning_rate = trial.suggest_categorical("learning_rate",
["constant", "adaptive", "invscaling"])

        clf = MLPClassifier(
            hidden_layer_sizes=hidden_layer_sizes,
            activation=activation,
            solver=solver,
            alpha=alpha,
            learning_rate=learning_rate,
            max_iter=1000,
            random_state=42
        )
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_val)

        return accuracy_score(y_val, y_pred)

    # Run Optuna search
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)

    print("\nBest Trial:")
    print(study.best_trial.params)

    # Train best model
    best_params = study.best_trial.params
    best_clf = MLPClassifier(**best_params, max_iter=1000,
random_state=42)
    best_clf.fit(X_train, y_train)

    y_pred = best_clf.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()

from sklearn.datasets import load_digits

digits = load_digits()

splits = [0.5, 0.4, 0.3, 0.2]
results = []
warnings.filterwarnings("ignore", category=UserWarning,
module="optuna")

for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_temp, y_train, y_temp = train_test_split(data,
digits.target, test_size=test_size, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)
    def objective(trial):

        hidden_layer_sizes =
trial.suggest_categorical("hidden_layer_sizes", [(100,), (200,),
(300,), (100,100), (200,100)])
        activation = trial.suggest_categorical("activation", ["relu",
"tanh", "logistic"])
        solver = trial.suggest_categorical("solver", ["adam", "sgd",
"lbfgs"])
        alpha = alpha = trial.suggest_float("alpha", 1e-5, 1e-2,
log=True)
        learning_rate = trial.suggest_categorical("learning_rate",
["constant", "adaptive", "invscaling"])

        clf = MLPClassifier(
            hidden_layer_sizes=hidden_layer_sizes,
            activation=activation,
            solver=solver,
            alpha=alpha,
            learning_rate=learning_rate,
            max_iter=1000,
            random_state=42
        )

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_val)

        return accuracy_score(y_val, y_pred)

    # Run Optuna search
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)

    print("\nBest Trial:")
    print(study.best_trial.params)

    # Train best model
    best_params = study.best_trial.params
    best_clf = MLPClassifier(**best_params, max_iter=1000,
random_state=42)
    best_clf.fit(X_train, y_train)

    y_pred = best_clf.predict(X_test)
    y_proba = best_clf.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, acc, prec, rec, f1])
    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        best_clf, X_train, y_train, cv=5, scoring="accuracy",
n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_clf.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_clf.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

results_df = pd.DataFrame(results, columns=["Test Size", "Accuracy",
"Precision", "Recall", "F1"])
display(results_df)

import matplotlib.pyplot as plt

results_df_t = results_df.drop(['Test Size'], axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size', labels=results_df['Test Size'])
plt.tight_layout()

plt.show()

print("\n=== PCA with Random Forest ===")

pca = PCA(n_components=10)
X_reduced = pca.fit_transform(data)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")

    X_train, X_temp, y_train, y_temp = train_test_split(X_reduced,
digits.target, test_size=test_size, random_state=42)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp,
test_size=0.5, random_state=42)

    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_val = scaler.transform(X_val)
    X_test = scaler.transform(X_test)

    def objective(trial):

        hidden_layer_sizes =
trial.suggest_categorical("hidden_layer_sizes", [(100,), (200,),
(300,), (100,100), (200,100)])
        activation = trial.suggest_categorical("activation", ["relu",
"tanh", "logistic"])
        solver = trial.suggest_categorical("solver", ["adam", "sgd",
"lbfgs"])
        alpha = alpha = trial.suggest_float("alpha", 1e-5, 1e-2,
log=True)
        learning_rate = trial.suggest_categorical("learning_rate",
["constant", "adaptive", "invscaling"])

        clf = MLPClassifier(
            hidden_layer_sizes=hidden_layer_sizes,
            activation=activation,
            solver=solver,
            alpha=alpha,
            learning_rate=learning_rate,
            max_iter=1000,
            random_state=42
        )

        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_val)

        return accuracy_score(y_val, y_pred)

    # Run Optuna search
    study = optuna.create_study(direction="maximize")
    study.optimize(objective, n_trials=50)

    print("\nBest Trial:")
    print(study.best_trial.params)

    # Train best model
    best_params = study.best_trial.params
    best_clf = MLPClassifier(**best_params, max_iter=1000,
random_state=42)
    best_clf.fit(X_train, y_train)

    y_pred = best_clf.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()
```

Results and Discussion

Wine Dataset

```
Best Trial:
{'hidden_layer_sizes': (300,), 'activation': 'relu', 'solver': 'sgd', 'alpha': 0.0008490330246413668, 'learning_rate': 'adaptive'}

Confusion Matrix
[[ 9  0  0]
 [ 0 11  0]
 [ 0  0  7]]
-------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00         9
     class_1       1.00      1.00      1.00        11
     class_2       1.00      1.00      1.00         7

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27
```

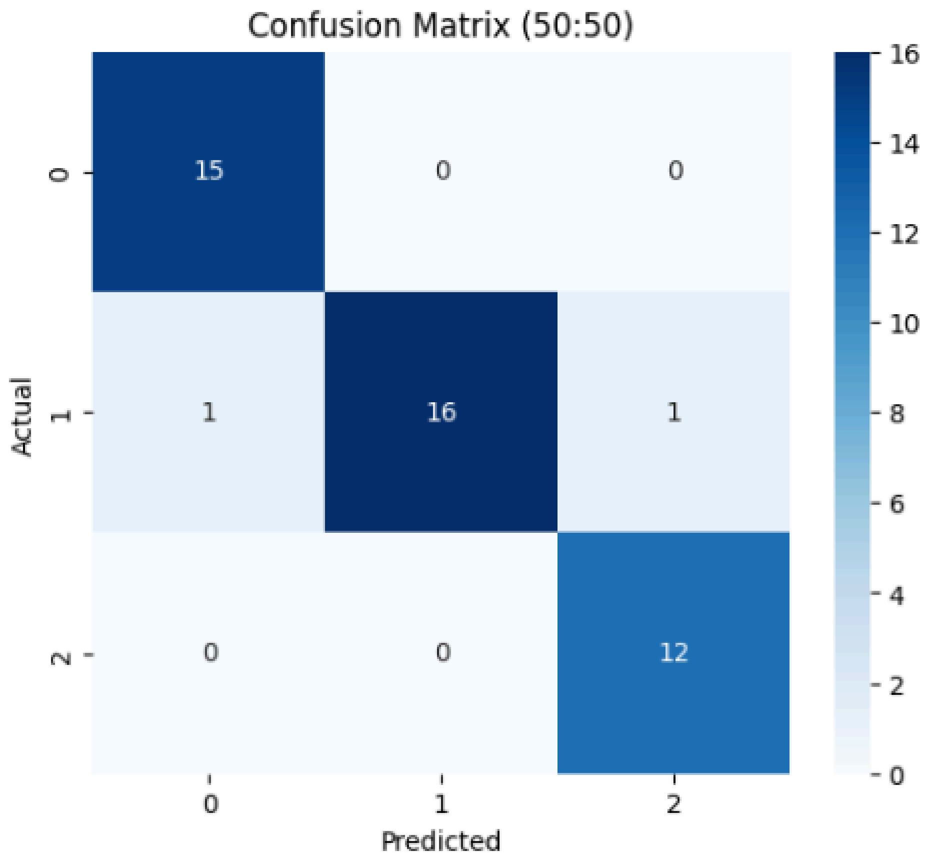


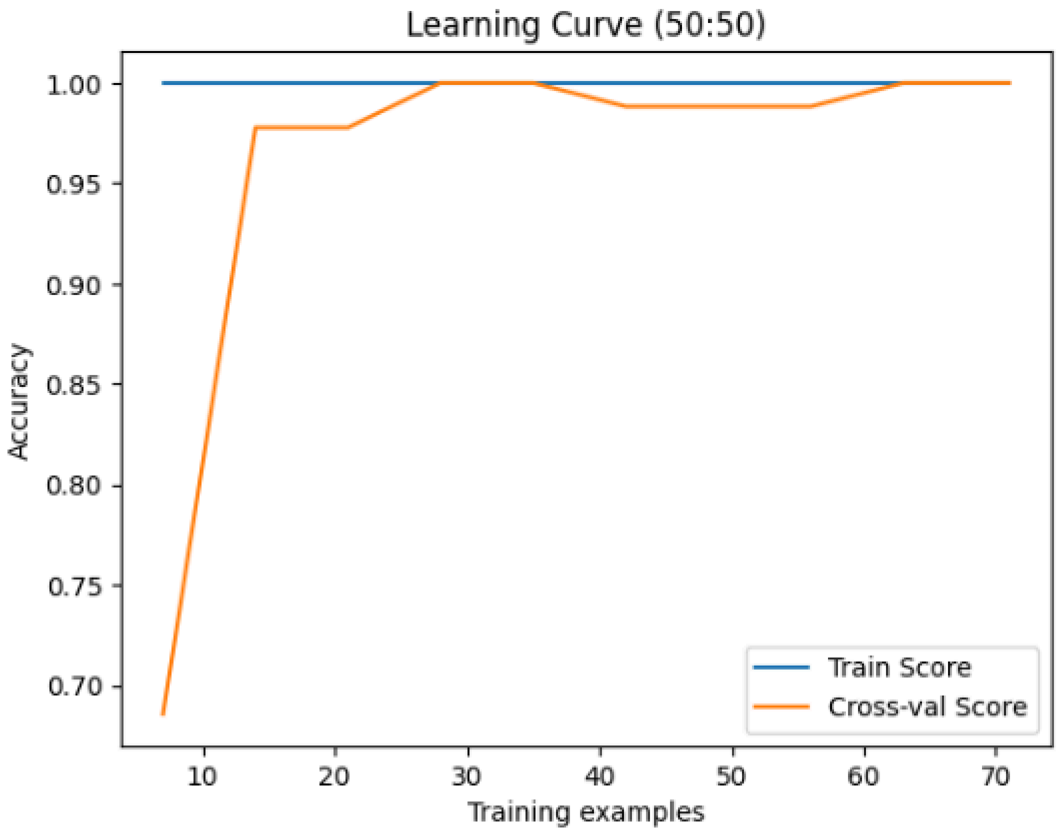
Comparison of different split sizes:

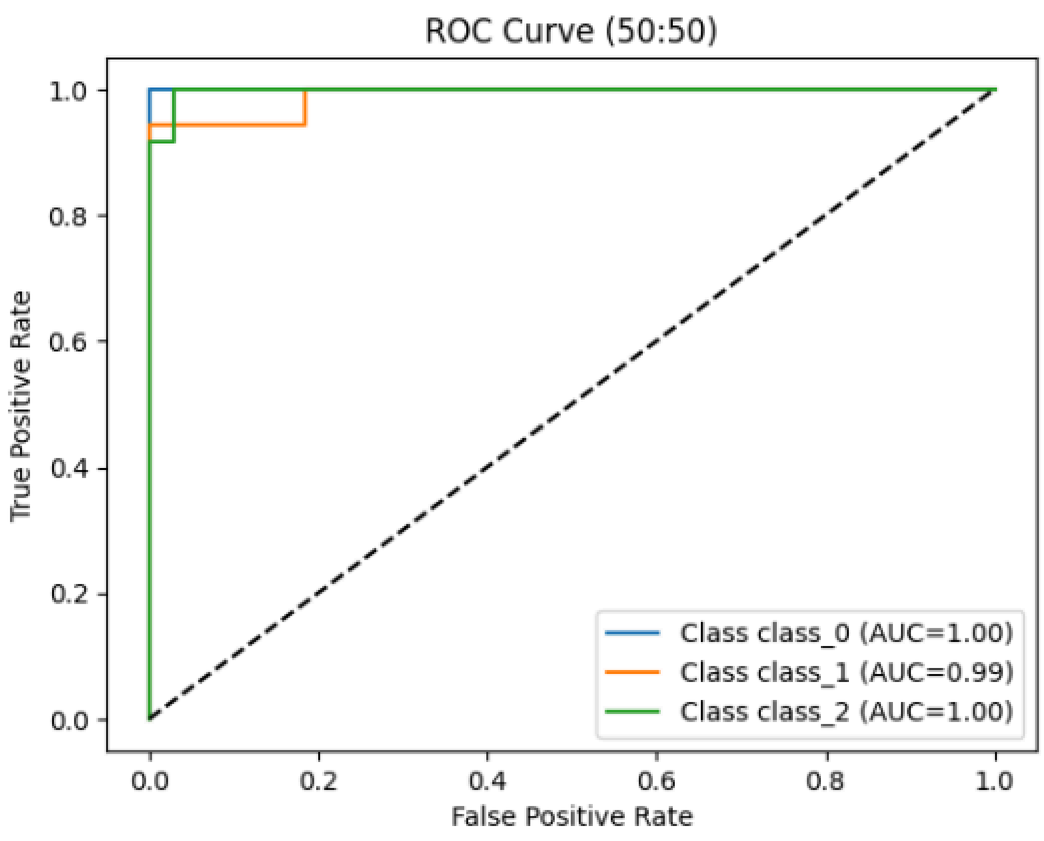
For each test size, the number of hidden layers, the activation functions, alpha and learning rate has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

Train Test Split (50:50)

```
Best Trial:
{'hidden_layer_sizes': (200, 100), 'activation': 'relu', 'solver': 'sgd', 'alpha': 1.727206677169179e-05, 'learning_rate': 'constant'}
              precision    recall  f1-score   support

     class_0       0.94      1.00      0.97        15
     class_1       1.00      0.89      0.94        18
     class_2       0.92      1.00      0.96        12

    accuracy                           0.96        45
   macro avg       0.95      0.96      0.96        45
weighted avg       0.96      0.96      0.96        45
```

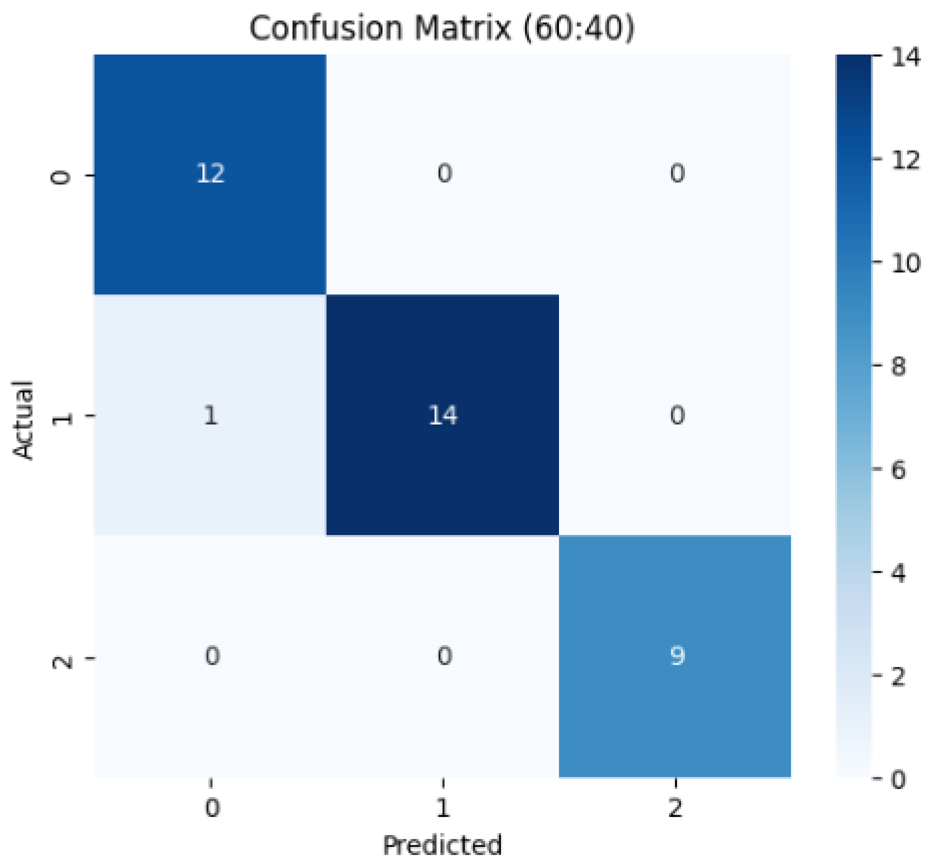

Confusion Matrix (50:50)
Actual
Predicted
0
1
2
15
0
0
1
16
1
0
0
12
16
14
12
10
8
6
4
2
0

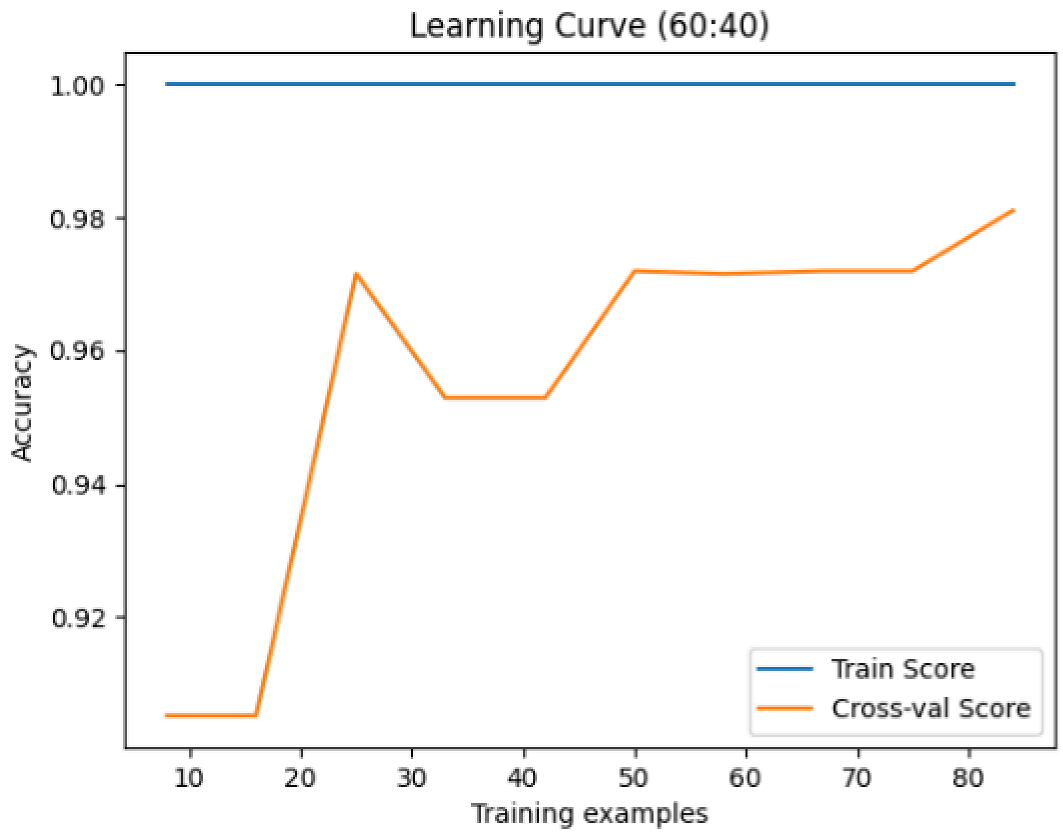

Learning Curve (50:50)
Accuracy
Training examples
1.00
0.95
0.90
0.85
0.80
0.75
0.70
10
20
30
40
50
60
70
Train Score
Cross-val Score

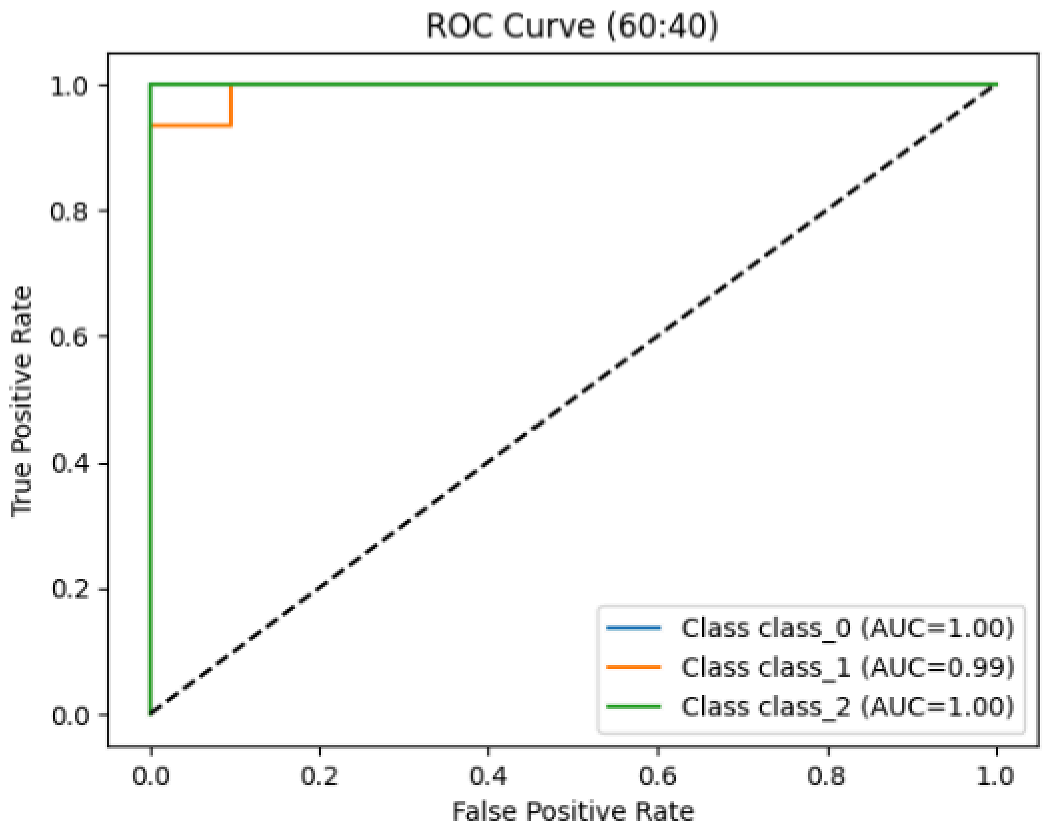


Train Test Split (60:40)

```
Best Trial:
{'hidden_layer_sizes': (100,), 'activation': 'tanh', 'solver': 'lbfgs', 'alpha': 2.385282645018547e-05, 'learning_rate': 'adaptive'}
              precision    recall  f1-score   support

     class_0       0.92      1.00      0.96        12
     class_1       1.00      0.93      0.97        15
     class_2       1.00      1.00      1.00         9

    accuracy                           0.97        36
   macro avg       0.97      0.98      0.98        36
weighted avg       0.97      0.97      0.97        36
```

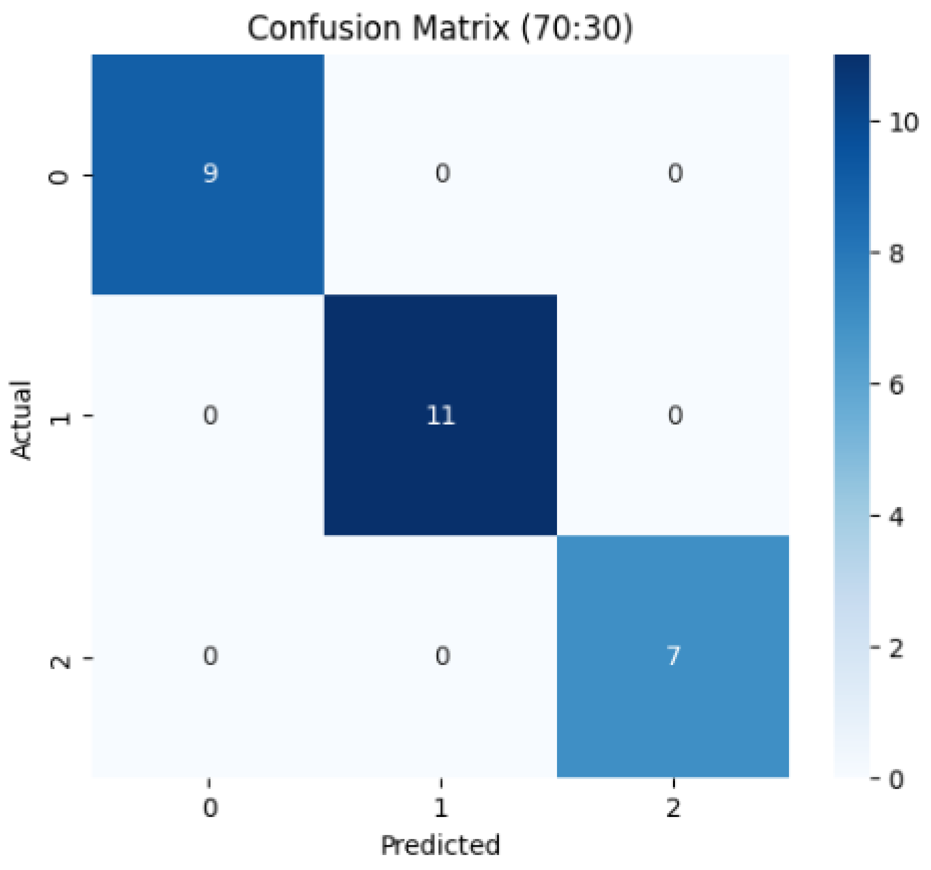

Confusion Matrix (60:40)
12
0
0
1
14
0
0
0
9
Actual
Predicted
0
1
2
0
1
2
14
12
10
8
6
4
2
0

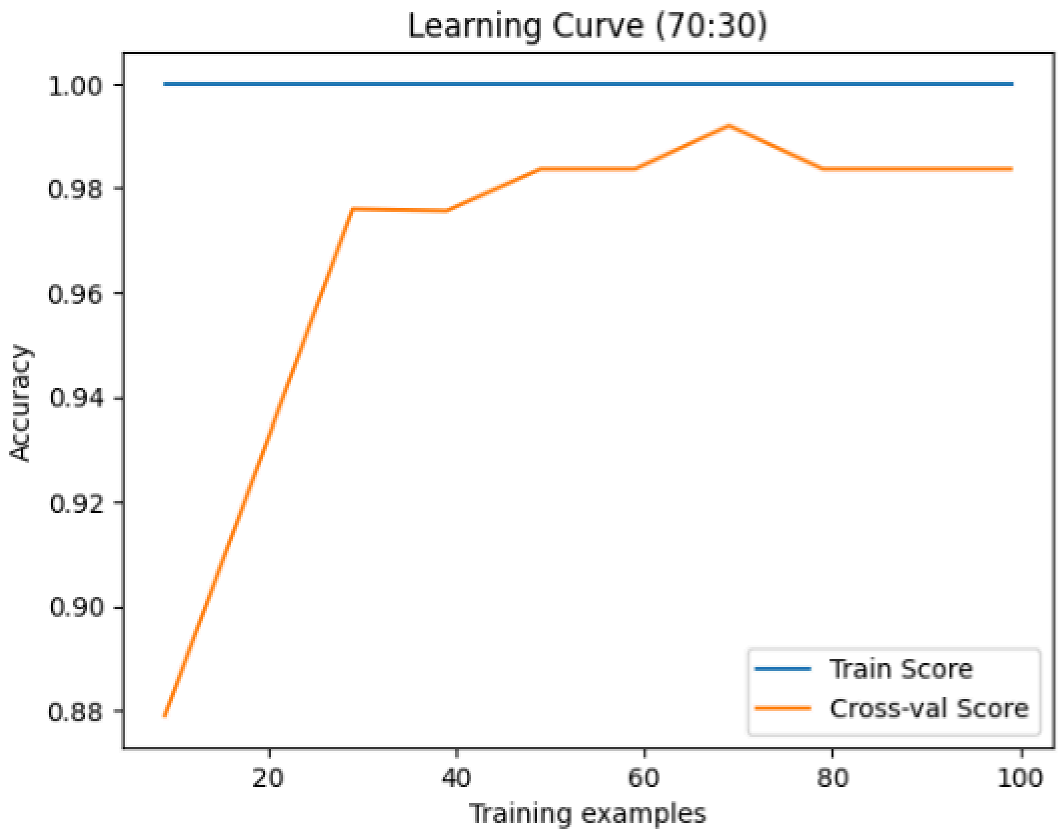

Learning Curve (60:40)
Accuracy
Training examples
1.00
0.98
0.96
0.94
0.92
10
20
30
40
50
60
70
80
Train Score
Cross-val Score

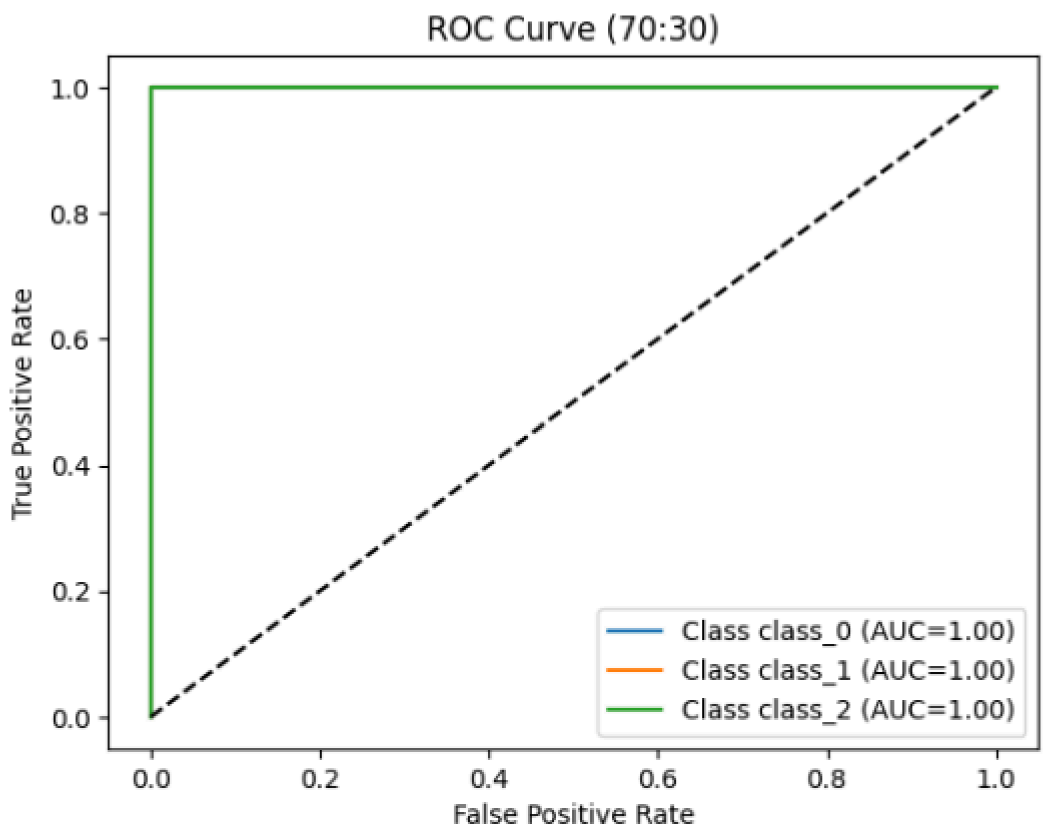


<u>Train Test Split (70:30)</u>

```
Best Trial:
{'hidden_layer_sizes': (200, 100), 'activation': 'relu', 'solver': 'adam', 'alpha': 0.0002989961023222637, 'learning_rate': 'invscaling'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00         9
     class_1       1.00      1.00      1.00        11
     class_2       1.00      1.00      1.00         7

    accuracy                           1.00        27
   macro avg       1.00      1.00      1.00        27
weighted avg       1.00      1.00      1.00        27
```

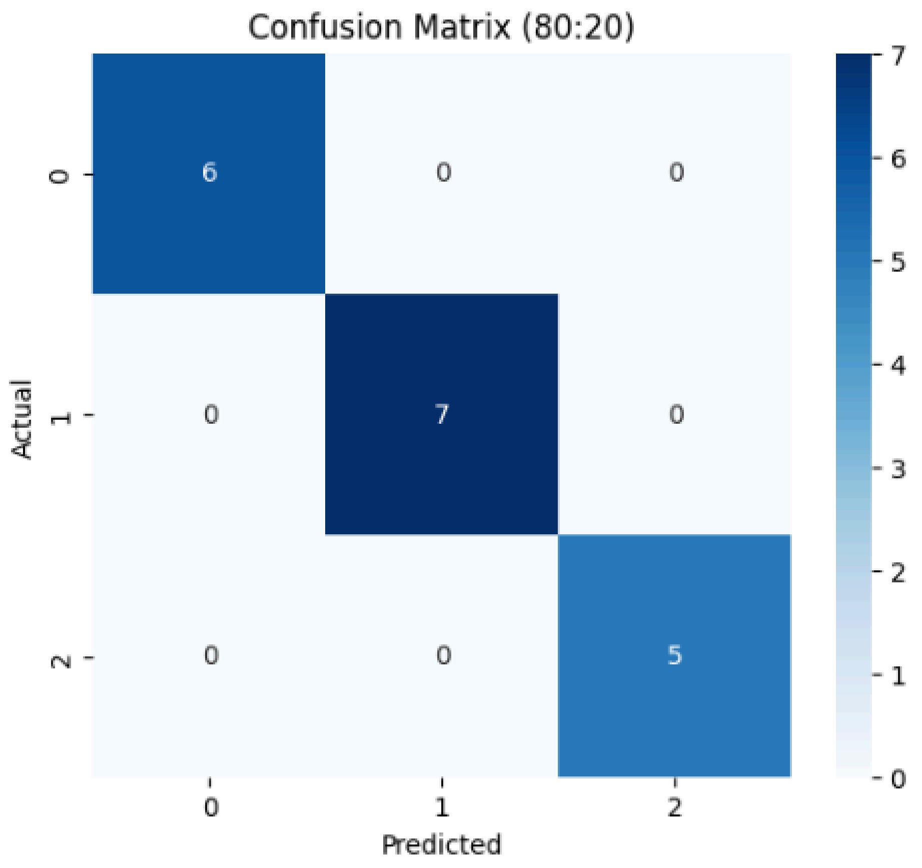

Confusion Matrix (70:30)
Actual
Predicted
0
1
2
9
0
0
0
11
0
0
0
7
10
8
6
4
2
0


Learning Curve (70:30)
Accuracy
Training examples
1.00
0.98
0.96
0.94
0.92
0.90
0.88
20
40
60
80
100
Train Score
Cross-val Score

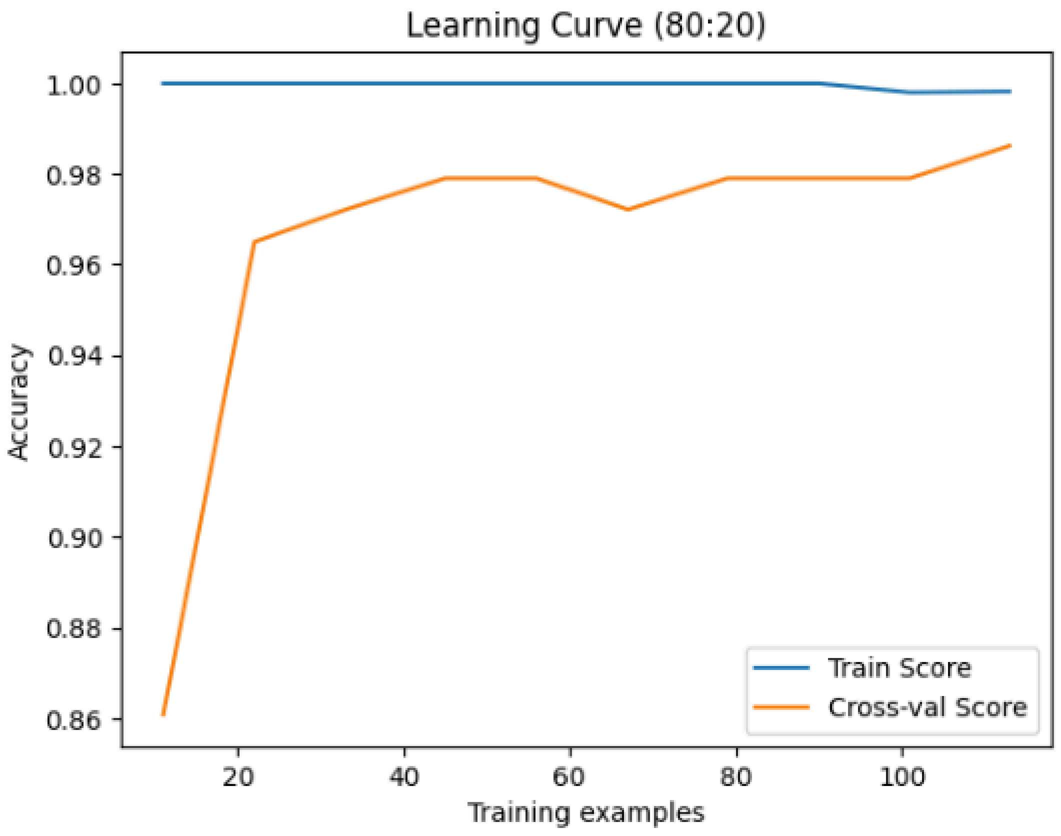


<u>Train Test Split (80:20)</u>

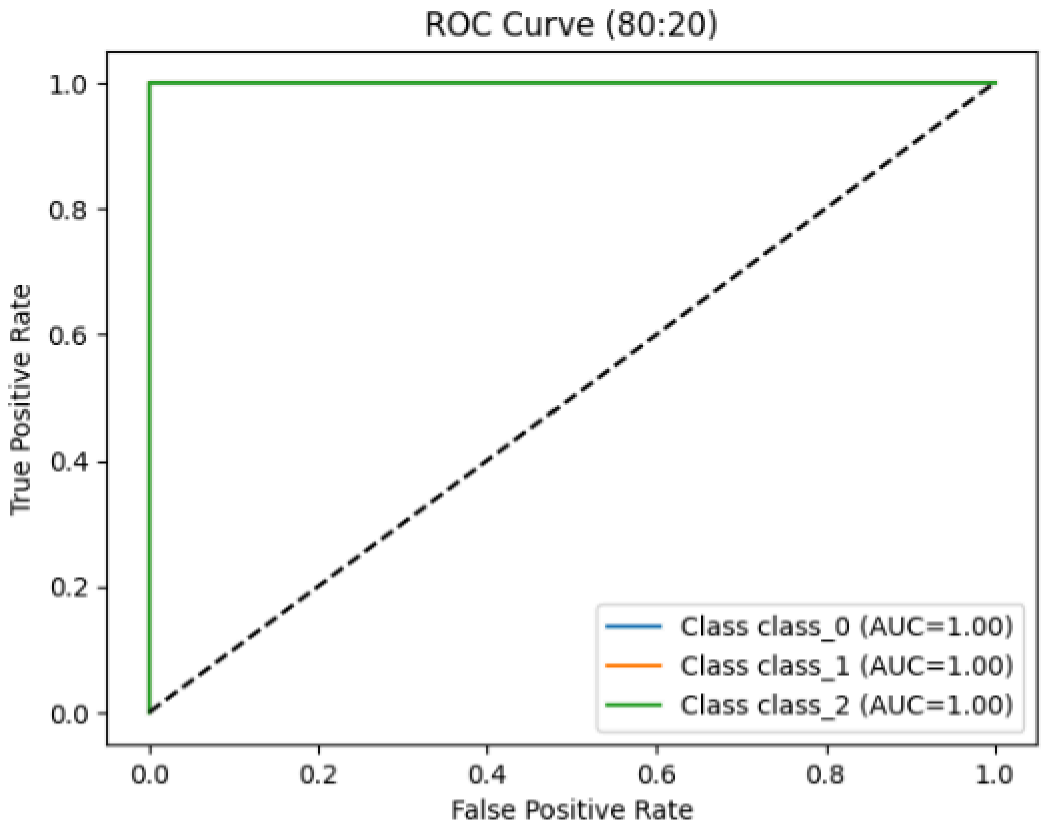
```
Best Trial:
{'hidden_layer_sizes': (100, 100), 'activation': 'tanh', 'solver': 'sgd', 'alpha': 0.001182434987246282, 'learning_rate': 'adaptive'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00         6
     class_1       1.00      1.00      1.00         7
     class_2       1.00      1.00      1.00         5

    accuracy                           1.00        18
   macro avg       1.00      1.00      1.00        18
weighted avg       1.00      1.00      1.00        18
```

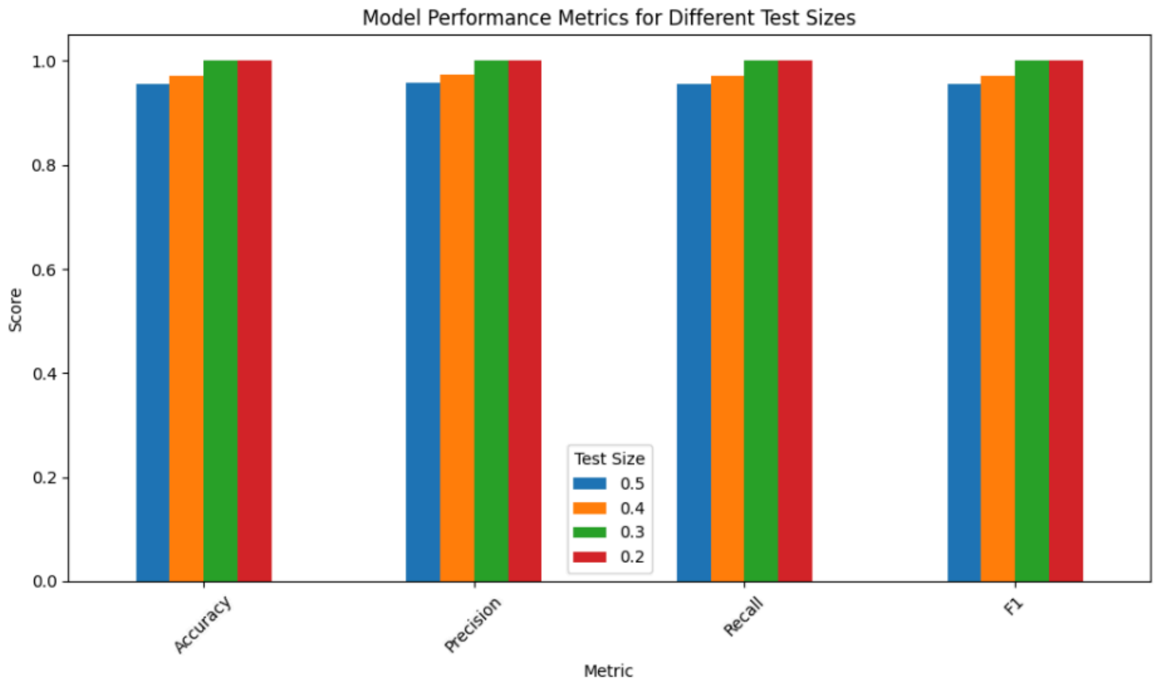


Confusion Matrix (80:20)
Actual
Predicted
0
1
2
6
7
5



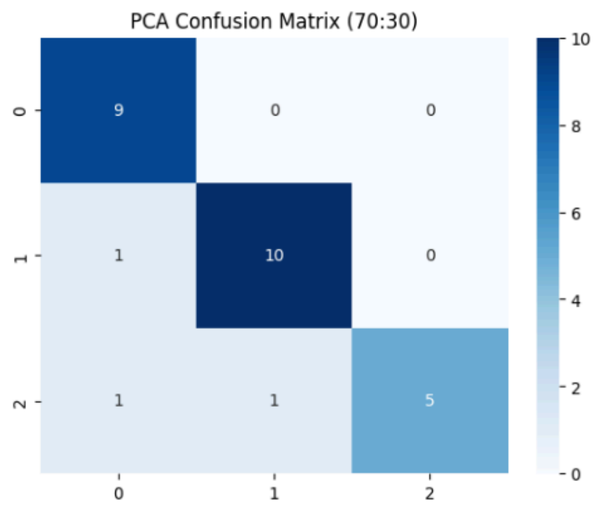
Learning Curve (80:20)
Accuracy
Training examples
Train Score
Cross-val Score





Principal Component Analysis (PCA) for feature dimensionality reduction

```
Best Trial:
{'hidden_layer_sizes': (300,), 'activation': 'tanh', 'solver': 'lbfgs', 'alpha': 0.0001793953775059501, 'learning_rate': 'constant'}
              precision    recall  f1-score   support

     class_0       0.82      1.00      0.90         9
     class_1       0.91      0.91      0.91        11
     class_2       1.00      0.71      0.83         7

    accuracy                           0.89        27
   macro avg       0.91      0.87      0.88        27
weighted avg       0.90      0.89      0.89        27
```



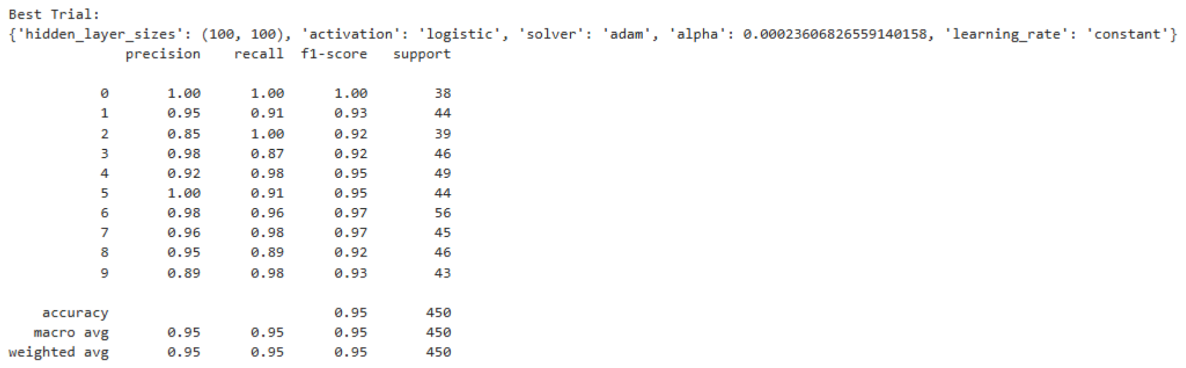
Digits Dataset

```
Confusion Matrix
[[51  0  0  0  1  0  1  0  0  0]
 [ 0 46  0  2  0  1  0  0  0  4]
 [ 0  0 52  1  0  0  0  0  0  0]
 [ 0  0  0 43  0  2  0  1  7  0]
 [ 1  0  0  0 53  0  0  0  0  3]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 53  1  0]
 [ 0  3  0  0  1  1  0  1 46  0]
 [ 0  0  0  0  0  2  0  1  1 51]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        53
           1       0.92      0.87      0.89        53
           2       1.00      0.98      0.99        53
           3       0.93      0.81      0.87        53
           4       0.96      0.93      0.95        57
           5       0.90      0.98      0.94        56
           6       0.96      0.98      0.97        54
           7       0.95      0.98      0.96        54
           8       0.84      0.88      0.86        52
           9       0.88      0.93      0.90        55

    accuracy                           0.93       540
   macro avg       0.93      0.93      0.93       540
weighted avg       0.93      0.93      0.93       540
```

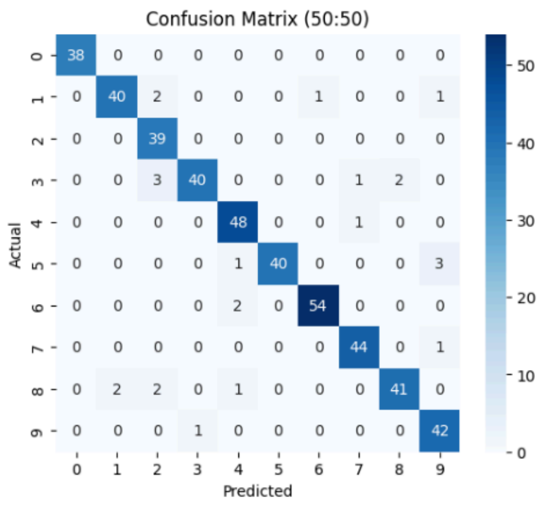
Comparison of different split sizes:

For each test size, the number of hidden layers, the activation functions, alpha and learning rate has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

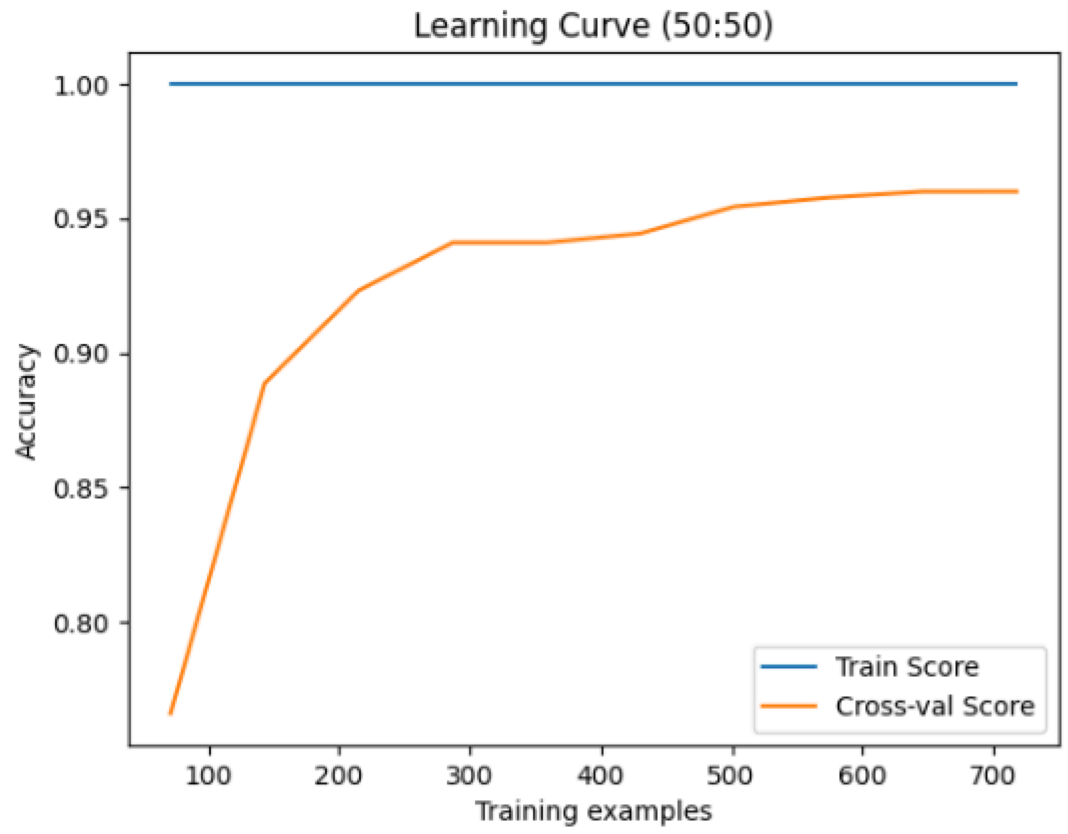
Train Test Split (50:50)

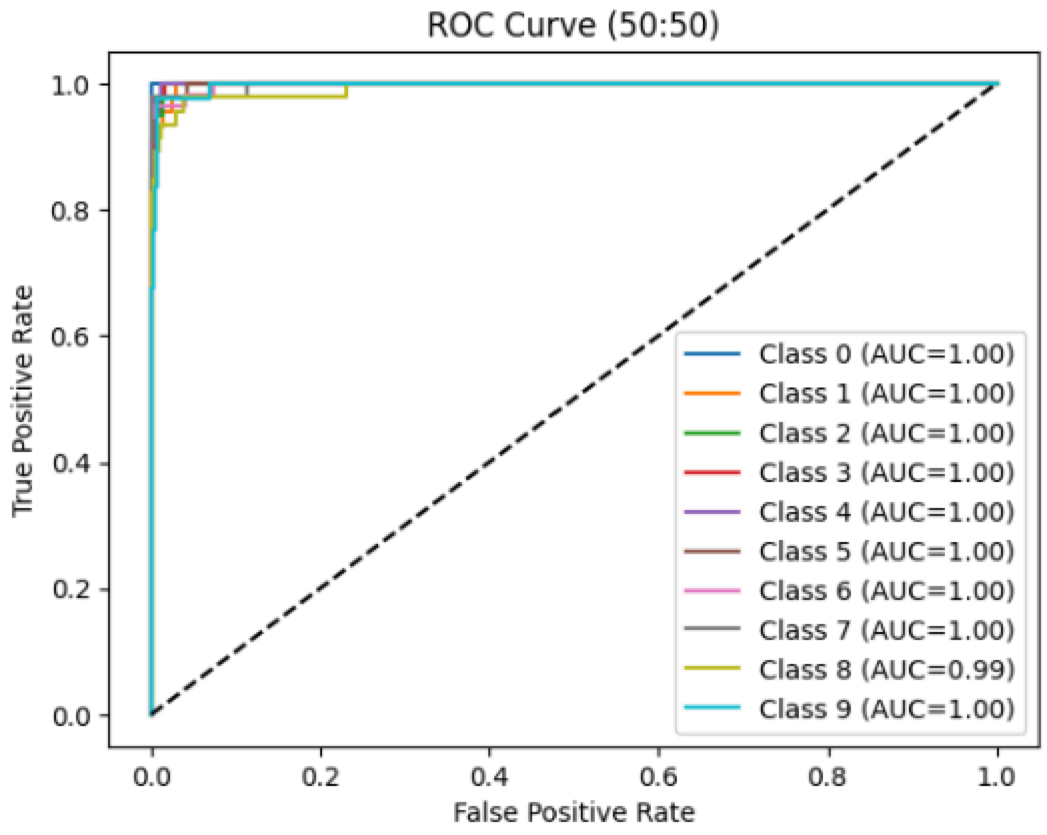


```
Best Trial:
{'hidden_layer_sizes': (100, 100), 'activation': 'logistic', 'solver': 'adam', 'alpha': 0.00023606826559140158, 'learning_rate': 'constant'}
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        38
           1       0.95      0.91      0.93        44
           2       0.85      1.00      0.92        39
           3       0.98      0.87      0.92        46
           4       0.92      0.98      0.95        49
           5       1.00      0.91      0.95        44
           6       0.98      0.96      0.97        56
           7       0.96      0.98      0.97        45
           8       0.95      0.89      0.92        46
           9       0.89      0.98      0.93        43

    accuracy                           0.95       450
   macro avg       0.95      0.95      0.95       450
weighted avg       0.95      0.95      0.95       450
```



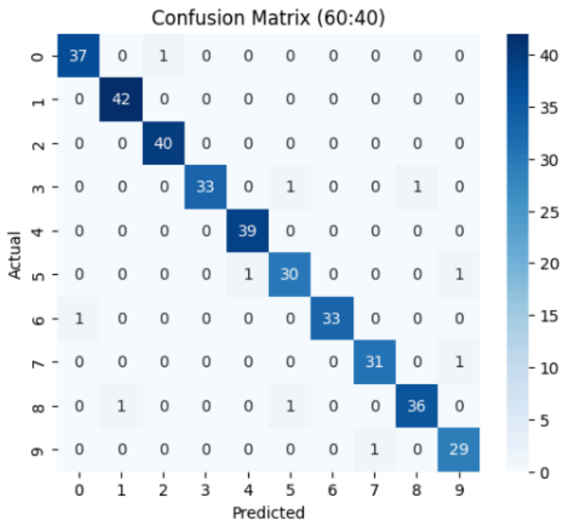
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	38	0	0	0	0	0	0	0	0	0
1	0	40	2	0	0	0	1	0	0	1
2	0	0	39	0	0	0	0	0	0	0
3	0	0	3	40	0	0	0	1	2	0
4	0	0	0	0	48	0	0	1	0	0
5	0	0	0	0	1	40	0	0	0	3
6	0	0	0	0	2	0	54	0	0	0
7	0	0	0	0	0	0	0	44	0	1
8	0	2	2	0	1	0	0	0	41	0
9	0	0	0	1	0	0	0	0	0	42

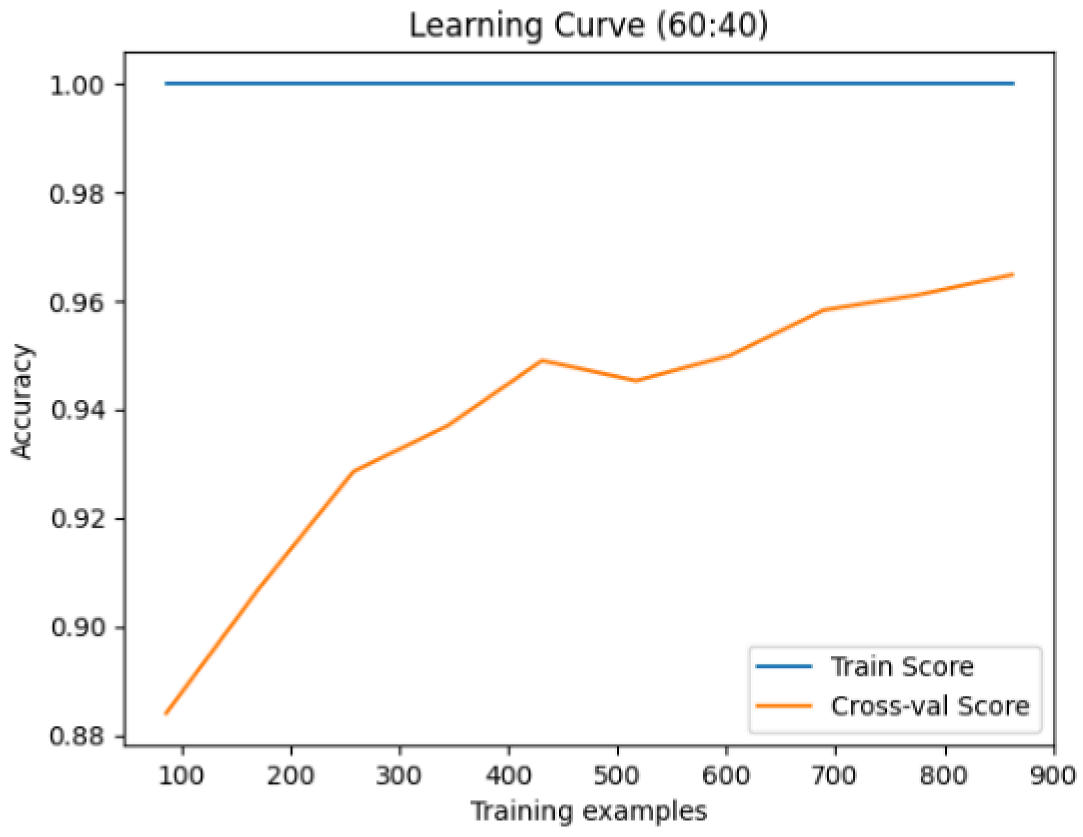


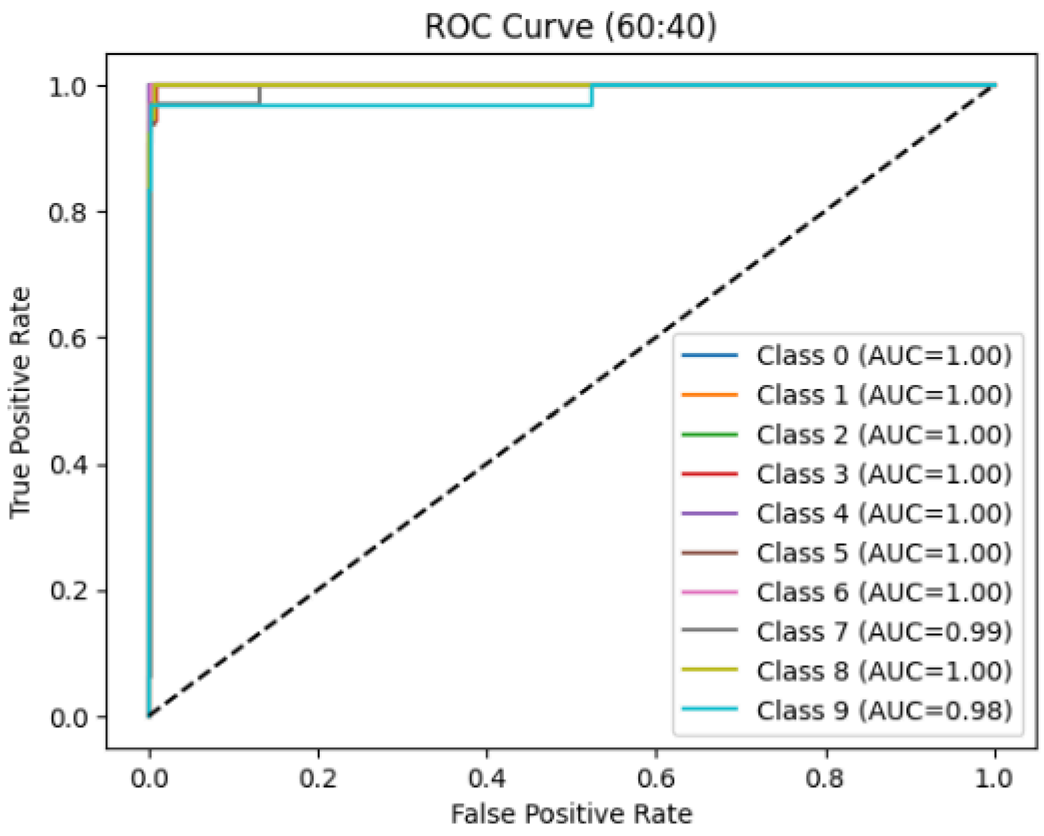


Train Test Split (60:40)

```
Best Trial:
{'hidden_layer_sizes': (200, 100), 'activation': 'relu', 'solver': 'adam', 'alpha': 0.00017121818249033981, 'learning_rate': 'adaptive'}
              precision    recall  f1-score   support

           0       0.97      0.97      0.97        38
           1       0.98      1.00      0.99        42
           2       0.98      1.00      0.99        40
           3       1.00      0.94      0.97        35
           4       0.97      1.00      0.99        39
           5       0.94      0.94      0.94        32
           6       1.00      0.97      0.99        34
           7       0.97      0.97      0.97        32
           8       0.97      0.95      0.96        38
           9       0.94      0.97      0.95        30

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360
```

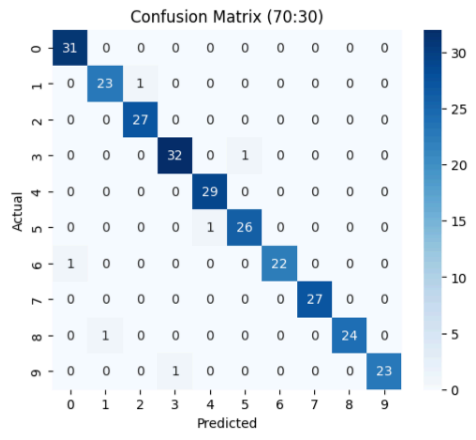


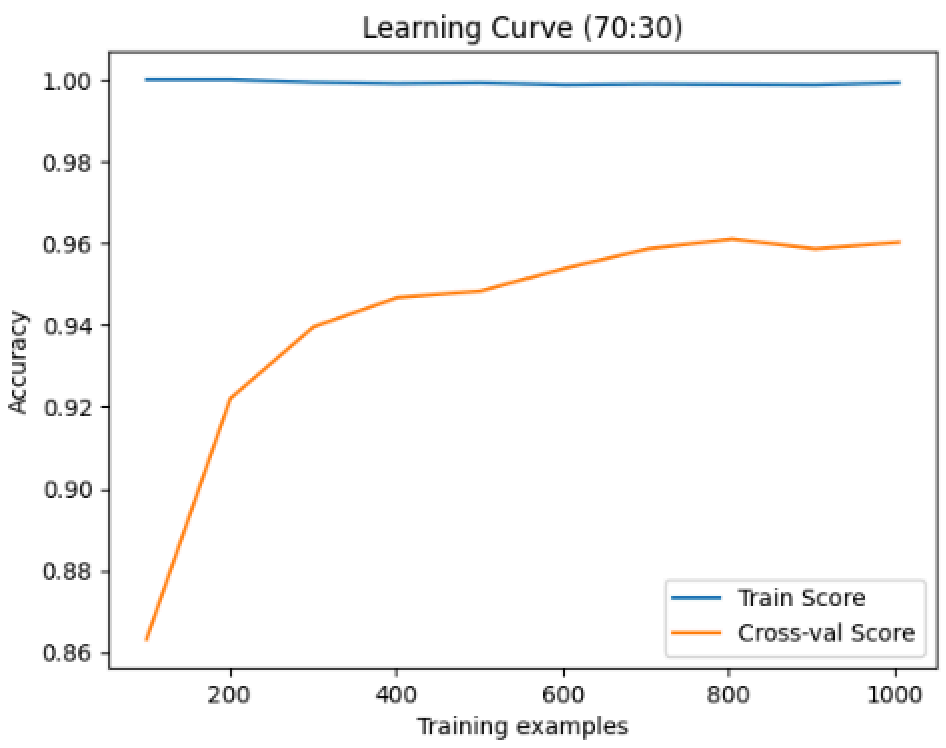


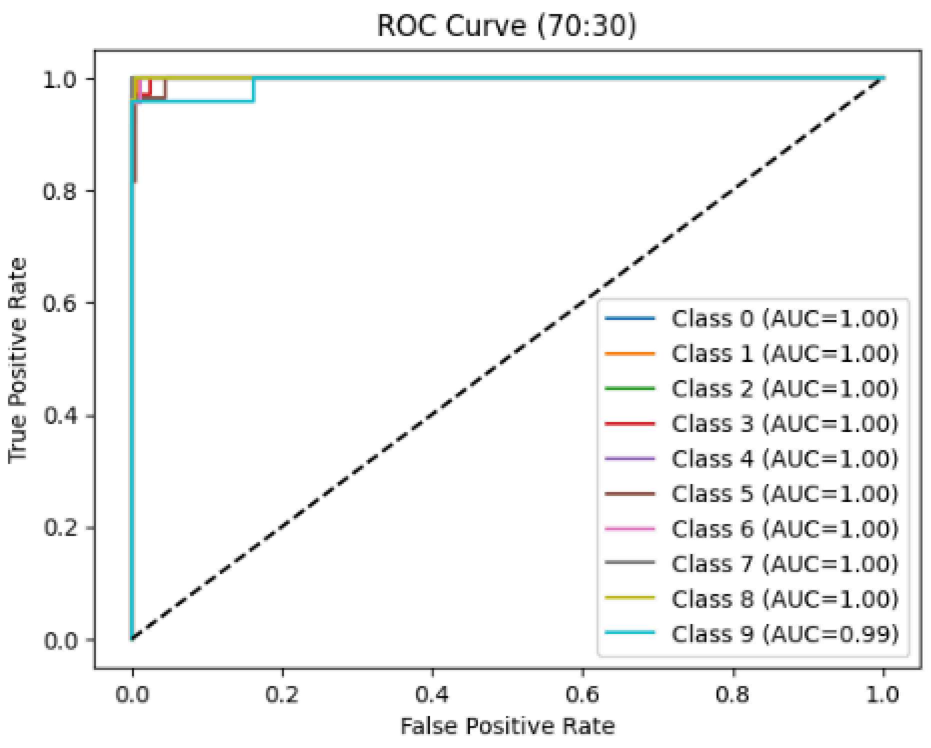


Train Test Split (70:30)

```
Best Trial:
{'hidden_layer_sizes': (100, 100), 'activation': 'relu', 'solver': 'sgd', 'alpha': 0.00307317100777546, 'learning_rate': 'adaptive'}
              precision    recall  f1-score   support

           0       0.97      1.00      0.98        31
           1       0.96      0.96      0.96        24
           2       0.96      1.00      0.98        27
           3       0.97      0.97      0.97        33
           4       0.97      1.00      0.98        29
           5       0.96      0.96      0.96        27
           6       1.00      0.96      0.98        23
           7       1.00      1.00      1.00        27
           8       1.00      0.96      0.98        25
           9       1.00      0.96      0.98        24

    accuracy                           0.98       270
   macro avg       0.98      0.98      0.98       270
weighted avg       0.98      0.98      0.98       270
```

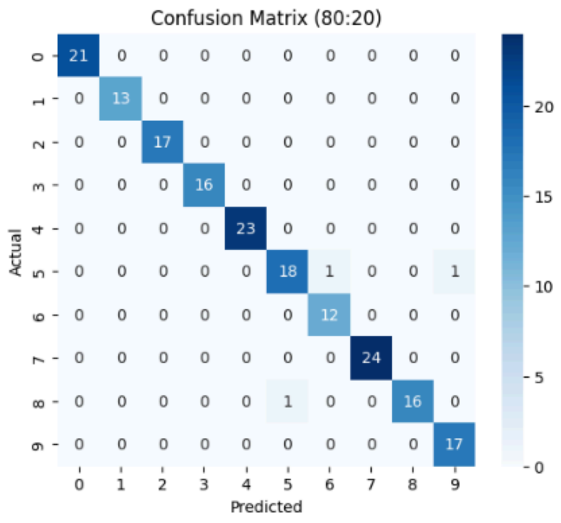


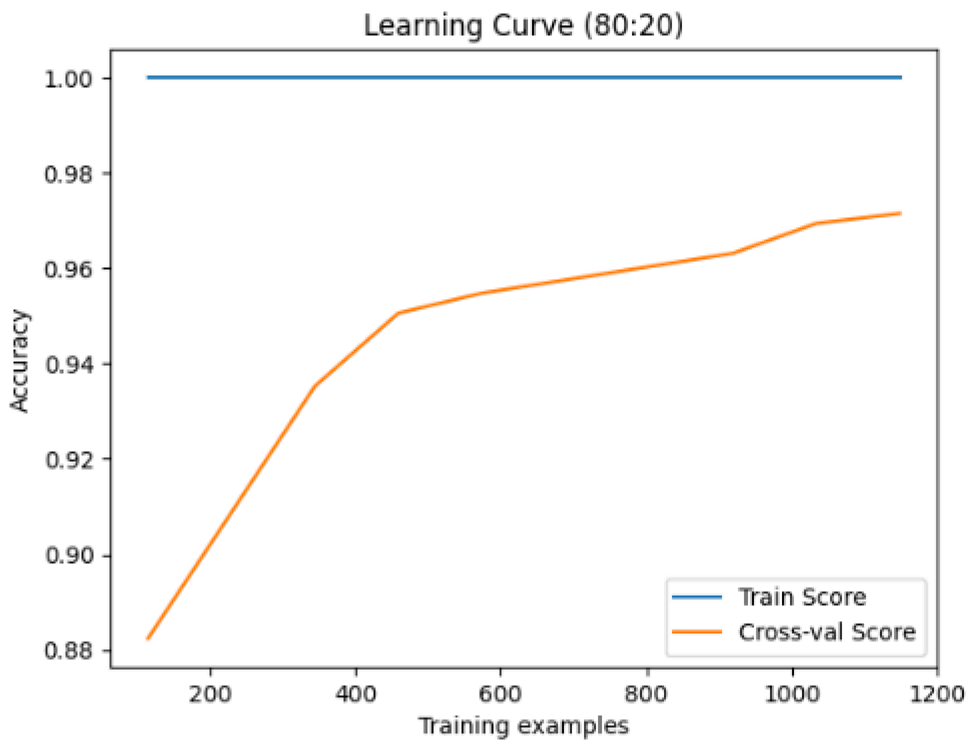




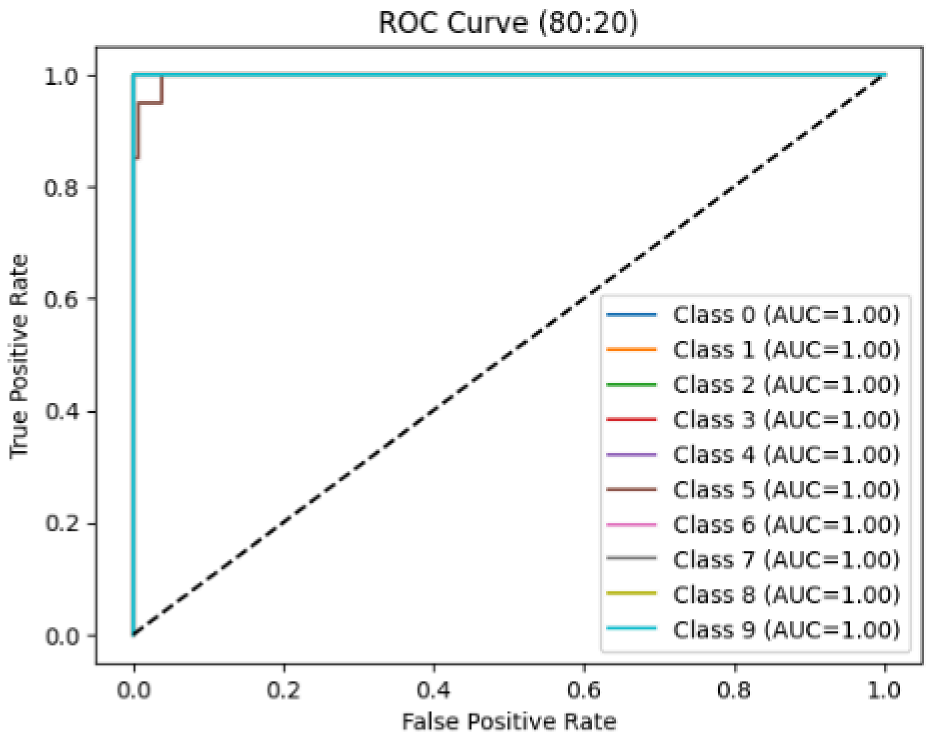
Train Test Split (80:20)

```
Best Trial:
{'hidden_layer_sizes': (100, 100), 'activation': 'tanh', 'solver': 'adam', 'alpha': 0.0008327393854627103, 'learning_rate': 'invscaling'}
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        21
           1       1.00      1.00      1.00        13
           2       1.00      1.00      1.00        17
           3       1.00      1.00      1.00        16
           4       1.00      1.00      1.00        23
           5       0.95      0.90      0.92        20
           6       0.92      1.00      0.96        12
           7       1.00      1.00      1.00        24
           8       1.00      0.94      0.97        17
           9       0.94      1.00      0.97        17

    accuracy                           0.98       180
   macro avg       0.98      0.98      0.98       180
weighted avg       0.98      0.98      0.98       180
```

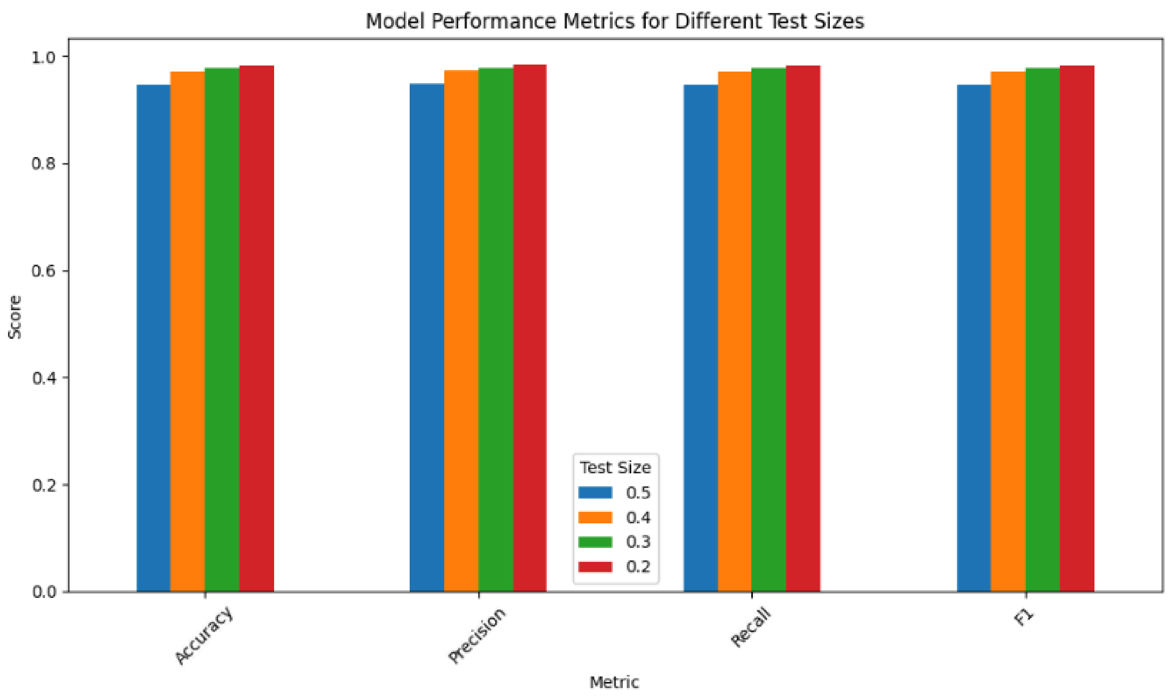




Learning Curve (80:20)
Accuracy
Training examples
Train Score
Cross-val Score
1.00
0.98
0.96
0.94
0.92
0.90
0.88
200
400
600
800
1000
1200

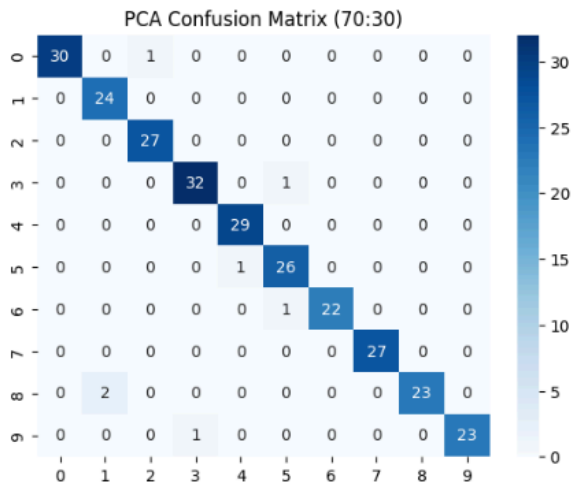


ROC Curve (80:20)
True Positive Rate
False Positive Rate
Class 0 (AUC=1.00)
Class 1 (AUC=1.00)
Class 2 (AUC=1.00)
Class 3 (AUC=1.00)
Class 4 (AUC=1.00)
Class 5 (AUC=1.00)
Class 6 (AUC=1.00)
Class 7 (AUC=1.00)
Class 8 (AUC=1.00)
Class 9 (AUC=1.00)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



Principal Component Analysis (PCA) for feature dimensionality reduction

```
Best Trial:
{'hidden_layer_sizes': (200, 100), 'activation': 'tanh', 'solver': 'adam', 'alpha': 0.0018002273106343608, 'learning_rate': 'constant'}
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        31
           1       0.92      1.00      0.96        24
           2       0.96      1.00      0.98        27
           3       0.97      0.97      0.97        33
           4       0.97      1.00      0.98        29
           5       0.93      0.96      0.95        27
           6       1.00      0.96      0.98        23
           7       1.00      1.00      1.00        27
           8       1.00      0.92      0.96        25
           9       1.00      0.96      0.98        24

    accuracy                           0.97       270
   macro avg       0.98      0.97      0.97       270
weighted avg       0.98      0.97      0.97       270
```



3. Random Forest classifier

Code:

```
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 sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score, precision_score, recall_score, f1_score, roc_curve, auc
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.model_selection import learning_curve
```

```
from sklearn.datasets import load_wine

wine = load_wine()
```

```
classifier = RandomForestClassifier()

classifier.fit(X_train, y_train)

# Evaluate on test set
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))
```

```
classifier = RandomForestClassifier()

param_grid = {
    'criterion': ['gini', 'entropy', 'log_loss'],
    # 'max_depth': [2, 3, 4, 5, 6],
    # 'min_samples_split': [2, 3, 4, 5],
    # 'min_samples_leaf': [1, 2, 3, 4, 5],
    # 'max_leaf_nodes': [2,3,4,5,6,7],
    # 'max_features': [None, 'sqrt', 'log2']
}

# Grid search
grid = GridSearchCV(classifier, param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

# Best model
best_model = grid.best_estimator_
print("Best Parameters:", grid.best_params_)

y_pred = best_model.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred,
target_names=wine.target_names))
```

```
splits = [0.5, 0.4, 0.3, 0.2]
results = []

for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=test_size, random_state=42, stratify=y
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    rf = RandomForestClassifier(random_state=42)

    param_grid = {
        'criterion': ['gini', 'entropy', 'log_loss'],
        # 'max_depth': [2, 3, 4, 5, 6],
        # 'min_samples_split': [2, 3, 4, 5],
        # 'min_samples_leaf': [1, 2, 3, 4, 5],
        # 'max_leaf_nodes': [2,3,4,5,6,7],
        # 'max_features': [None, 'sqrt', 'log2']
    }

    grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)
    y_proba = best_model.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, acc, prec, rec, f1])

    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        rf, X_train, y_train, cv=5, scoring="accuracy", n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
```

```python
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_model.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_model.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

results_df = pd.DataFrame(results, columns=["Test Size", "Accuracy",
"Precision", "Recall", "F1"])
display(results_df)
```

```python
import matplotlib.pyplot as plt

results_df_t = results_df.drop('Test Size', axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size')
plt.tight_layout()

plt.show()
```

```python
print("\n=== PCA with Random Forest ===")
```

```
pca = PCA(n_components=10)
X_reduced = pca.fit_transform(X)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")
    X_train, X_test, y_train, y_test = train_test_split(
        X_reduced, y, test_size=test_size, random_state=42, stratify=y
    )

    rf = RandomForestClassifier(random_state=42)

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    param_grid = {
        'criterion': ['gini', 'entropy', 'log_loss'],
        # 'max_depth': [2, 3, 4, 5, 6],
        # 'min_samples_split': [2, 3, 4, 5],
        # 'min_samples_leaf': [1, 2, 3, 4, 5],
        # 'max_leaf_nodes': [2,3,4,5,6,7],
        # 'max_features': [None, 'sqrt', 'log2']
    }

    grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()
```

```
from sklearn.datasets import load_digits

digits = load_digits()
```

```
# Classifier
classifier = RandomForestClassifier(criterion='gini', max_depth=20,
max_features='sqrt')

# Fit
classifier.fit(X_train, y_train)

# Predict
y_pred = classifier.predict(X_test)

print("\nConfusion Matrix")
print(confusion_matrix(y_test, y_pred))

print("-------------------------------------------------------")
print("Classification Report")
print(classification_report(y_test, y_pred))
```

```
splits = [0.5, 0.4, 0.3, 0.2]
results = []

for test_size in splits:
    print(f"\n=== Train-Test Split:
{int((1-test_size)*100)}:{int(test_size*100)} ===")

    X_train, X_test, y_train, y_test = train_test_split(
        data, digits.target, test_size=test_size, shuffle=False
    )

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    # Train
    rf = RandomForestClassifier(random_state=42)

    param_grid = {
        'criterion': ['gini', 'entropy', 'log_loss'],
        # 'max_depth': [2, 3, 4, 5, 6],
        # 'min_samples_split': [2, 3, 4, 5],
        # 'min_samples_leaf': [1, 2, 3, 4, 5],
        # 'max_leaf_nodes': [2,3,4,5,6,7],
        # 'max_features': [None, 'sqrt', 'log2']
    }

    grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)
    y_proba = best_model.predict_proba(X_test)

    # Metrics
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted")
    rec = recall_score(y_test, y_pred, average="weighted")
    f1 = f1_score(y_test, y_pred, average="weighted")
    results.append([test_size, acc, prec, rec, f1])

    print(classification_report(y_test, y_pred))

    # Confusion Matrix Heatmap
    plt.figure(figsize=(6,5))
    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Predicted"); plt.ylabel("Actual")
    plt.show()

    # Learning Curve
    train_sizes, train_scores, test_scores = learning_curve(
        rf, X_train, y_train, cv=5, scoring="accuracy", n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10)
    )
    plt.figure()
    plt.plot(train_sizes, np.mean(train_scores, axis=1), label="Train
Score")
    plt.plot(train_sizes, np.mean(test_scores, axis=1),
label="Cross-val Score")
```

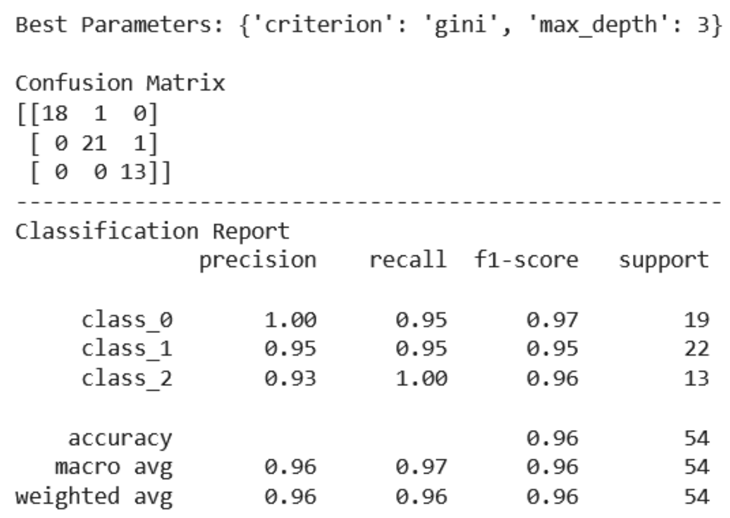
```
    plt.title(f"Learning Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("Training examples"); plt.ylabel("Accuracy")
    plt.legend(); plt.show()

    # ROC Curve
    fpr, tpr, roc_auc = {}, {}, {}
    for i, cls in enumerate(best_model.classes_):
        fpr[i], tpr[i], _ = roc_curve(y_test == cls, y_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    plt.figure()
    for i, cls in enumerate(best_model.classes_):
        plt.plot(fpr[i], tpr[i], label=f"Class {cls}
(AUC={roc_auc[i]:.2f})")
    plt.plot([0,1],[0,1],"k--")
    plt.title(f"ROC Curve
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.xlabel("False Positive Rate"); plt.ylabel("True Positive Rate")
    plt.legend(); plt.show()

# Summary table
import pandas as pd
results_df = pd.DataFrame(results, columns=["Test Size", "Accuracy",
"Precision", "Recall", "F1"])
display(results_df)
```

```
import matplotlib.pyplot as plt

results_df_t = results_df.drop('Test Size', axis=1)
results_df_t = results_df_t.T

results_df_t.plot(kind='bar', figsize=(10, 6))

plt.title('Model Performance Metrics for Different Test Sizes')
plt.xlabel('Metric')
plt.ylabel('Score')
plt.xticks(rotation=45)
plt.legend(title='Test Size')
plt.tight_layout()

plt.show()
```

```
pca = PCA(n_components=10)
X_reduced = pca.fit_transform(data)

for test_size in splits:
    print(f"\n--- PCA {int((1-test_size)*100)}:{int(test_size*100)}
---")
    X_train, X_test, y_train, y_test = train_test_split(
        X_reduced, digits.target, test_size=test_size, shuffle=False
    )

    rf = RandomForestClassifier(random_state=42)

    sc = StandardScaler()
    X_train = sc.fit_transform(X_train)
    X_test = sc.transform(X_test)

    param_grid = {
        'criterion': ['gini', 'entropy', 'log_loss'],
        # 'max_depth': [2, 3, 4, 5, 6],
        # 'min_samples_split': [2, 3, 4, 5],
        # 'min_samples_leaf': [1, 2, 3, 4, 5],
        # 'max_leaf_nodes': [2,3,4,5,6,7],
        # 'max_features': [None, 'sqrt', 'log2']
    }

    grid = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    print("Best Parameters:", grid.best_params_)

    y_pred = best_model.predict(X_test)

    print(classification_report(y_test, y_pred))

    sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt="d",
cmap="Blues")
    plt.title(f"PCA Confusion Matrix
({int((1-test_size)*100)}:{int(test_size*100)})")
    plt.show()
```

Results and Discussion

Wine Dataset



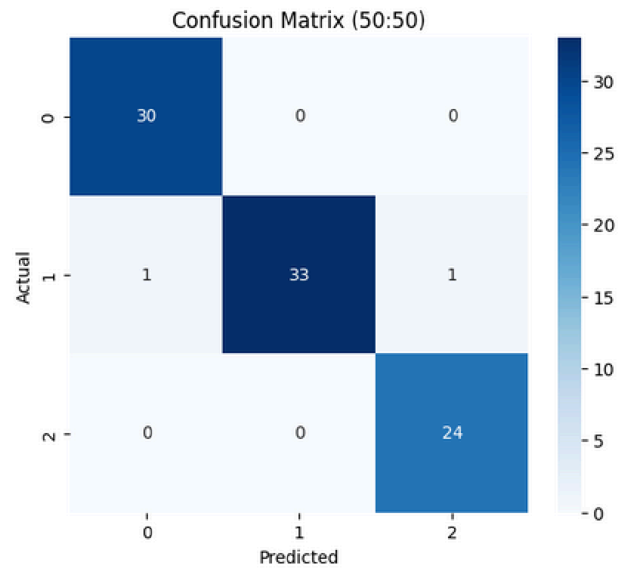
```
Best Parameters: {'criterion': 'gini', 'max_depth': 3}

Confusion Matrix
[[18  1  0]
 [ 0 21  1]
 [ 0  0 13]]
-------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

     class_0       1.00      0.95      0.97        19
     class_1       0.95      0.95      0.95        22
     class_2       0.93      1.00      0.96        13

    accuracy                           0.96        54
   macro avg       0.96      0.97      0.96        54
weighted avg       0.96      0.96      0.96        54
```

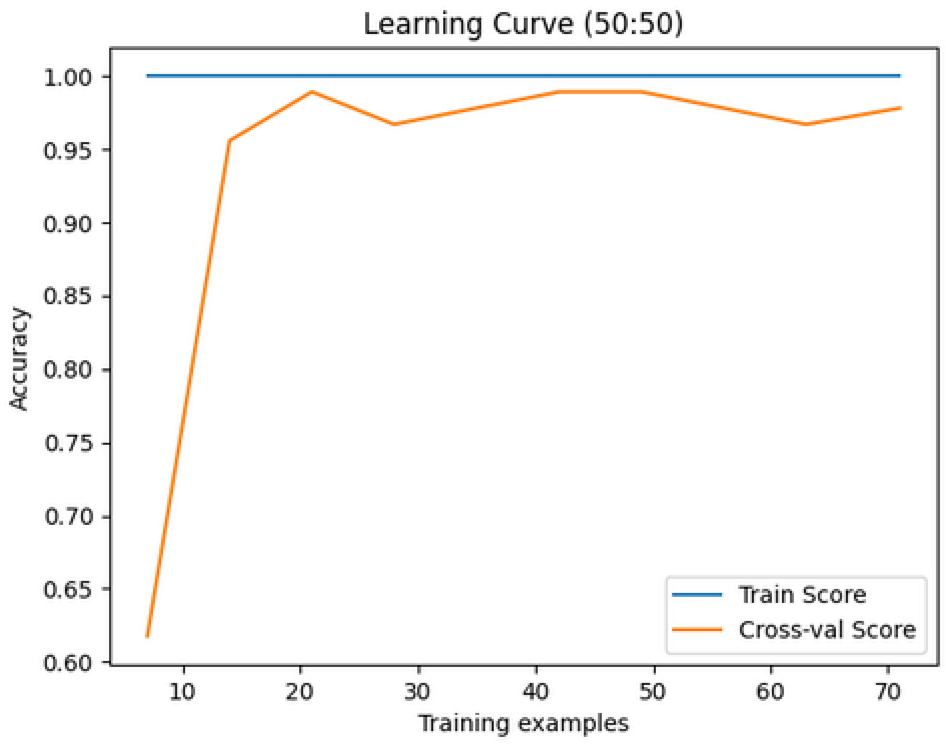
Comparison of different split sizes:

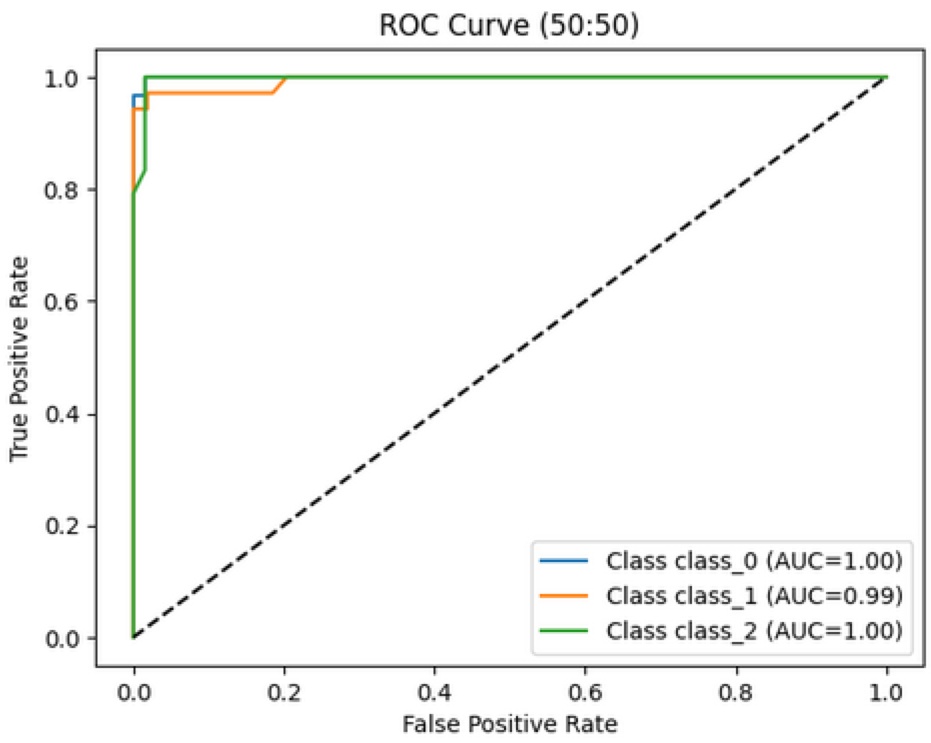
For each test size, the criteria, max_depth, max_features etc has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

Train Test Split (50:50)

```
=== Train-Test Split: 50:50 ===
Best Parameters: {'criterion': 'gini'}
              precision    recall  f1-score   support

     class_0       0.97      1.00      0.98        30
     class_1       1.00      0.94      0.97        35
     class_2       0.96      1.00      0.98        24

    accuracy                           0.98        89
   macro avg       0.98      0.98      0.98        89
weighted avg       0.98      0.98      0.98        89
```

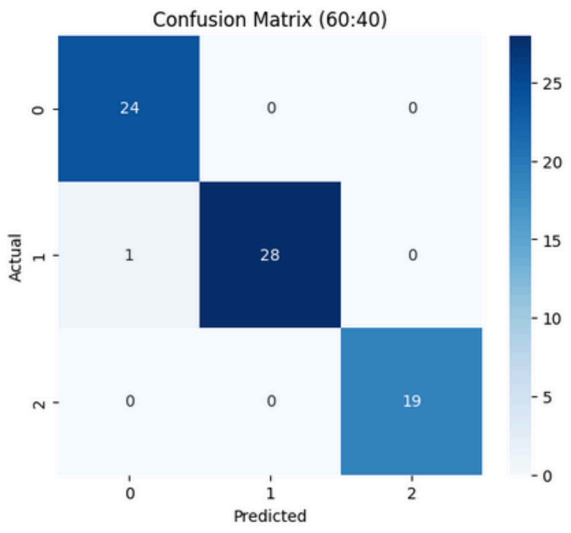


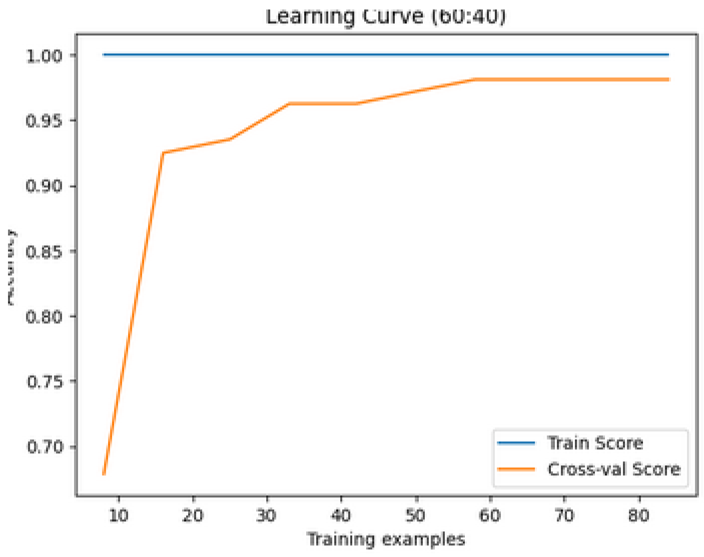


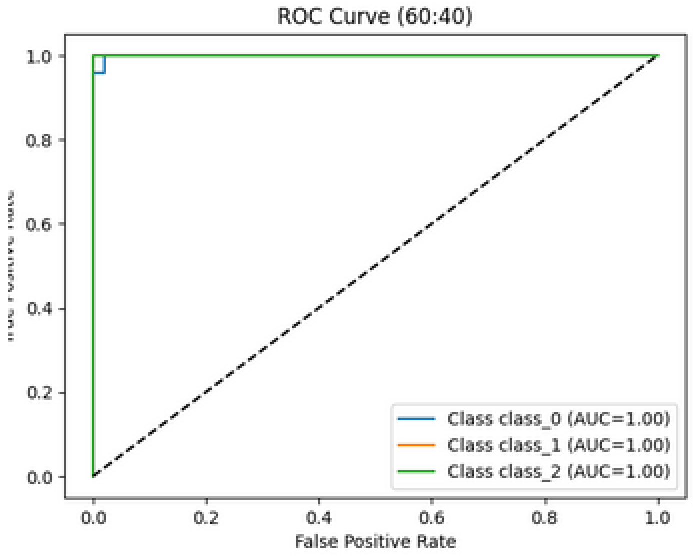


<u>Train Test Split (60:40)</u>

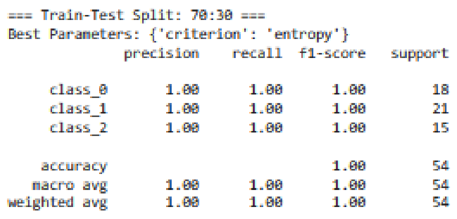
```
=== Train-Test Split: 60:40 ===
Best Parameters: {'criterion': 'gini'}
              precision    recall  f1-score   support

     class_0       0.96      1.00      0.98        24
     class_1       1.00      0.97      0.98        29
     class_2       1.00      1.00      1.00        19

    accuracy                           0.99        72
   macro avg       0.99      0.99      0.99        72
weighted avg       0.99      0.99      0.99        72
```



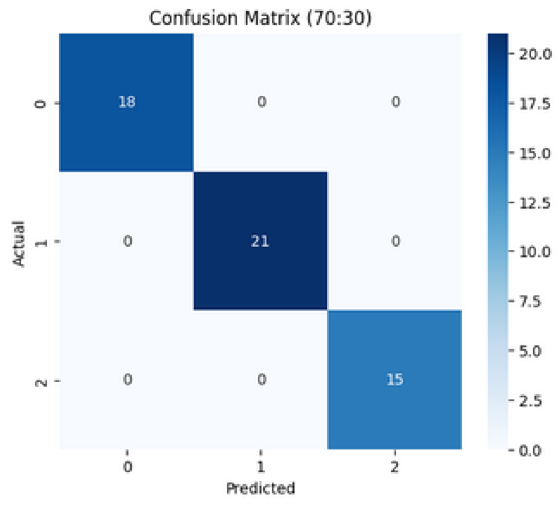


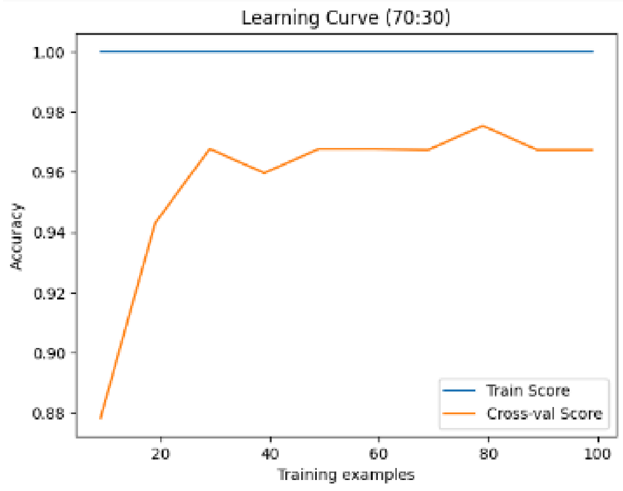


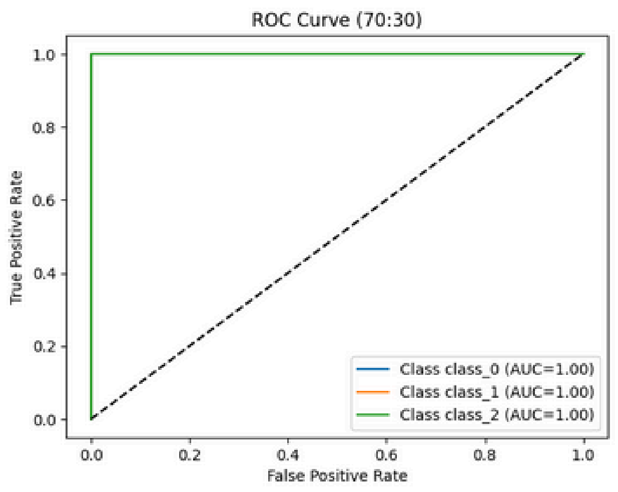
<u>**Train Test Split (70:30)**</u>



```
=== Train-Test Split: 70:30 ===
Best Parameters: {'criterion': 'entropy'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        18
     class_1       1.00      1.00      1.00        21
     class_2       1.00      1.00      1.00        15

    accuracy                           1.00        54
   macro avg       1.00      1.00      1.00        54
weighted avg       1.00      1.00      1.00        54
```

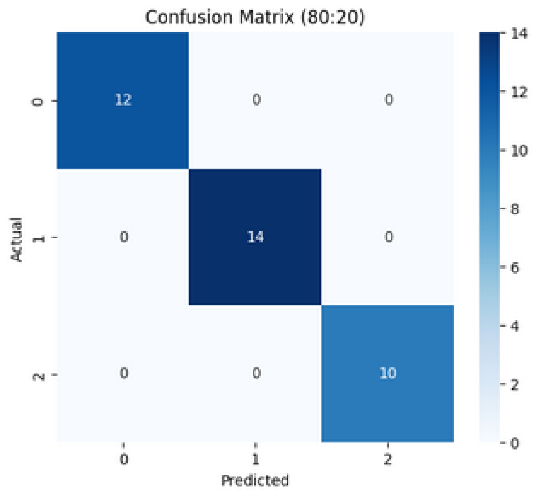


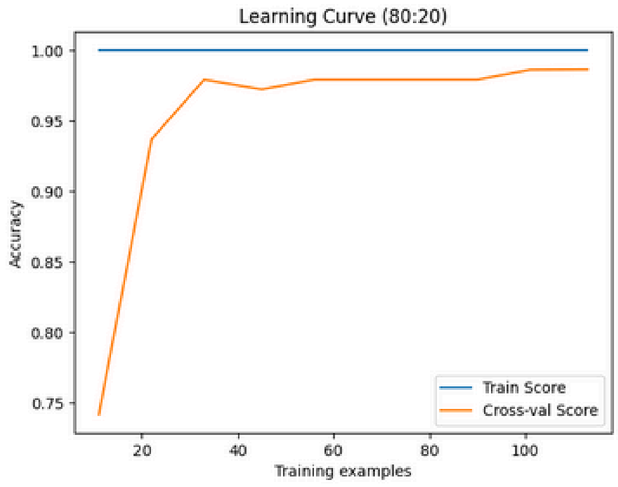


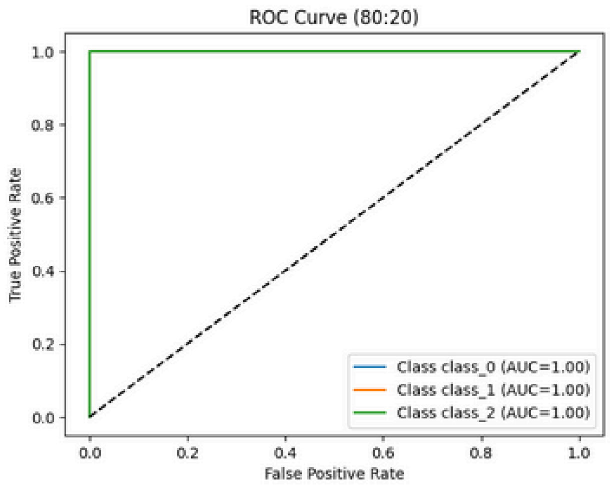


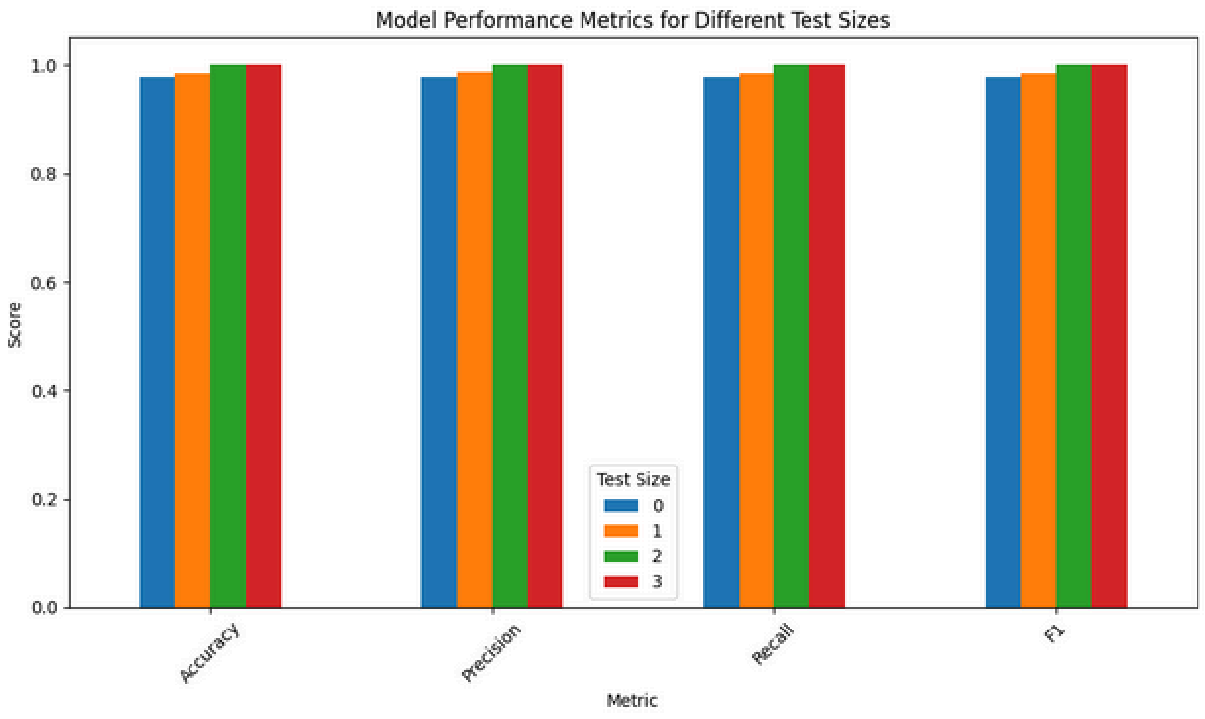
<u>Train Test Split (80:20)</u>

```
=== Train-Test Split: 80:20 ===
Best Parameters: {'criterion': 'gini'}
              precision    recall  f1-score   support

     class_0       1.00      1.00      1.00        12
     class_1       1.00      1.00      1.00        14
     class_2       1.00      1.00      1.00        10

    accuracy                           1.00        36
   macro avg       1.00      1.00      1.00        36
weighted avg       1.00      1.00      1.00        36
```



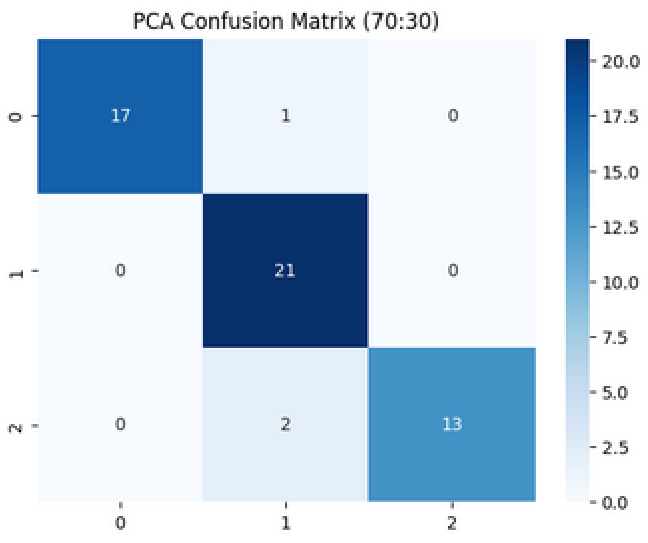




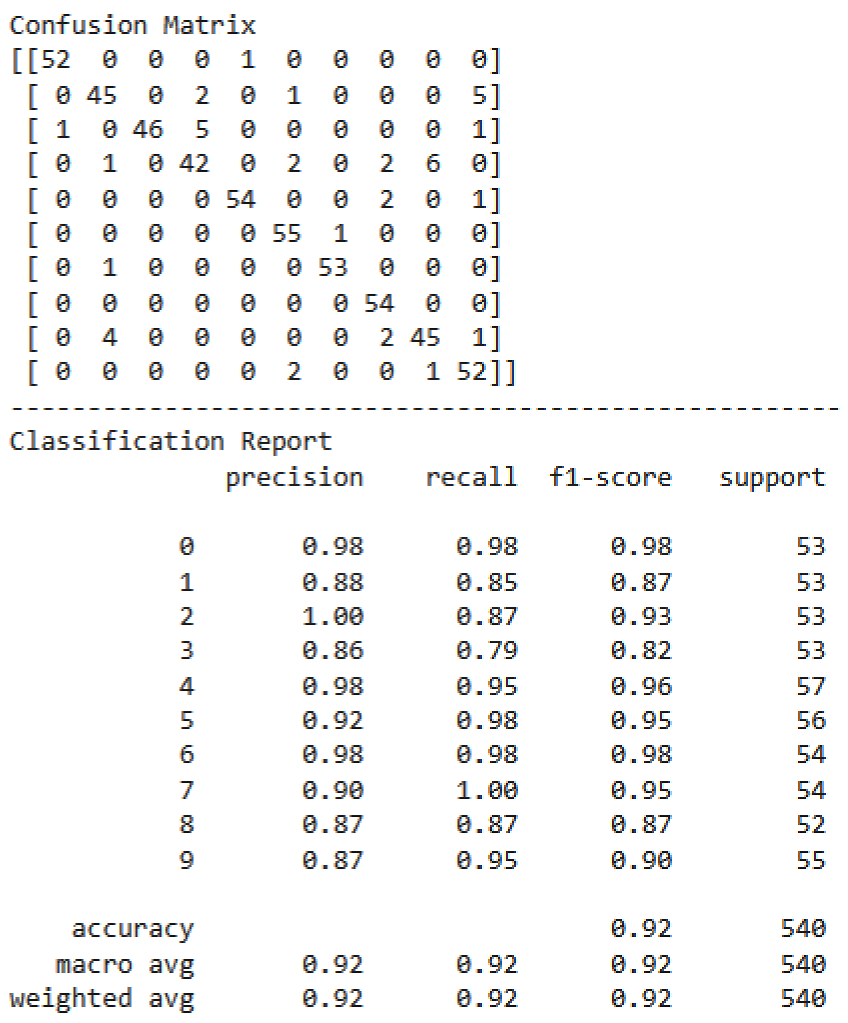


Principal Component Analysis (PCA) for feature dimensionality reduction

```
--- PCA 70:30 ---
Best Parameters: {'criterion': 'gini'}
              precision    recall  f1-score   support

     class_0       1.00      0.94      0.97        18
     class_1       0.88      1.00      0.93        21
     class_2       1.00      0.87      0.93        15

    accuracy                           0.94        54
   macro avg       0.96      0.94      0.94        54
weighted avg       0.95      0.94      0.94        54
```



Digits Dataset



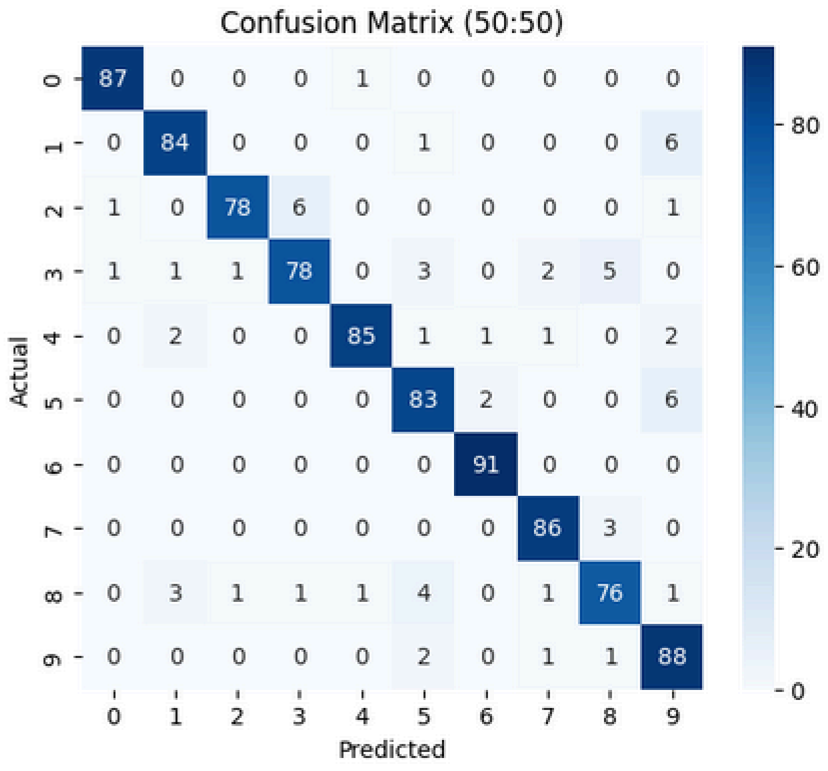
```
Confusion Matrix
[[52  0  0  0  1  0  0  0  0  0]
 [ 0 45  0  2  0  1  0  0  0  5]
 [ 1  0 46  5  0  0  0  0  0  1]
 [ 0  1  0 42  0  2  0  2  6  0]
 [ 0  0  0  0 54  0  0  2  0  1]
 [ 0  0  0  0  0 55  1  0  0  0]
 [ 0  1  0  0  0  0 53  0  0  0]
 [ 0  0  0  0  0  0  0 54  0  0]
 [ 0  4  0  0  0  0  0  2 45  1]
 [ 0  0  0  0  0  2  0  0  1 52]]
------------------------------------------------------
Classification Report
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        53
           1       0.88      0.85      0.87        53
           2       1.00      0.87      0.93        53
           3       0.86      0.79      0.82        53
           4       0.98      0.95      0.96        57
           5       0.92      0.98      0.95        56
           6       0.98      0.98      0.98        54
           7       0.90      1.00      0.95        54
           8       0.87      0.87      0.87        52
           9       0.87      0.95      0.90        55

    accuracy                           0.92       540
   macro avg       0.92      0.92      0.92       540
weighted avg       0.92      0.92      0.92       540
```

Comparison of different split sizes:

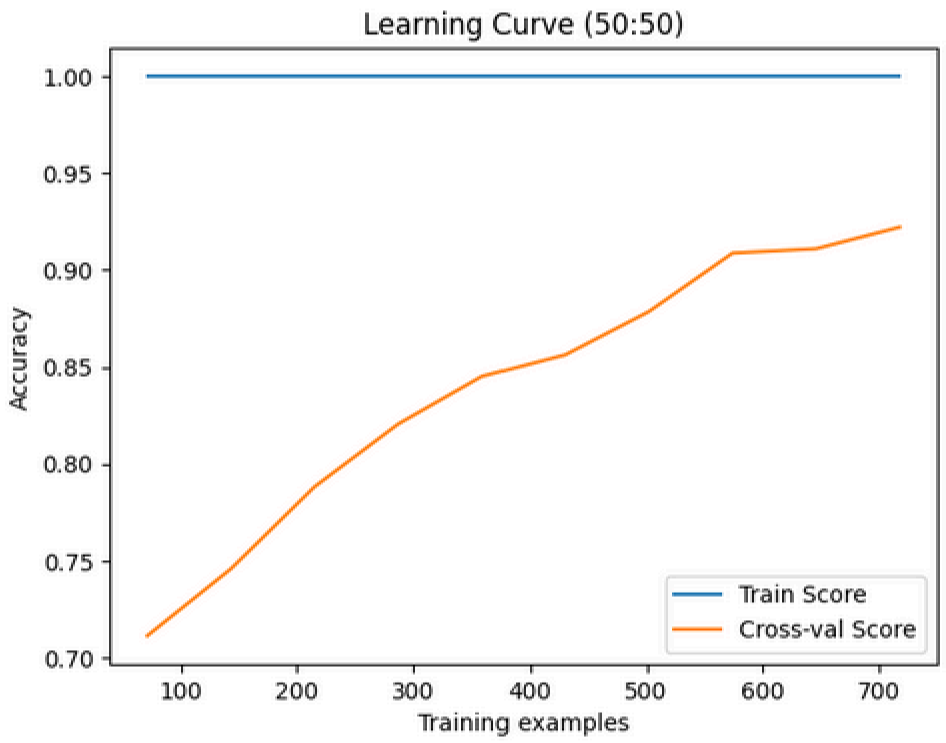
For each test size, the number of hidden layers, the activation functions, alpha and learning rate has been searched and applied. The confusion matrix, Learning Curve and ROC Curve have been generated for each.

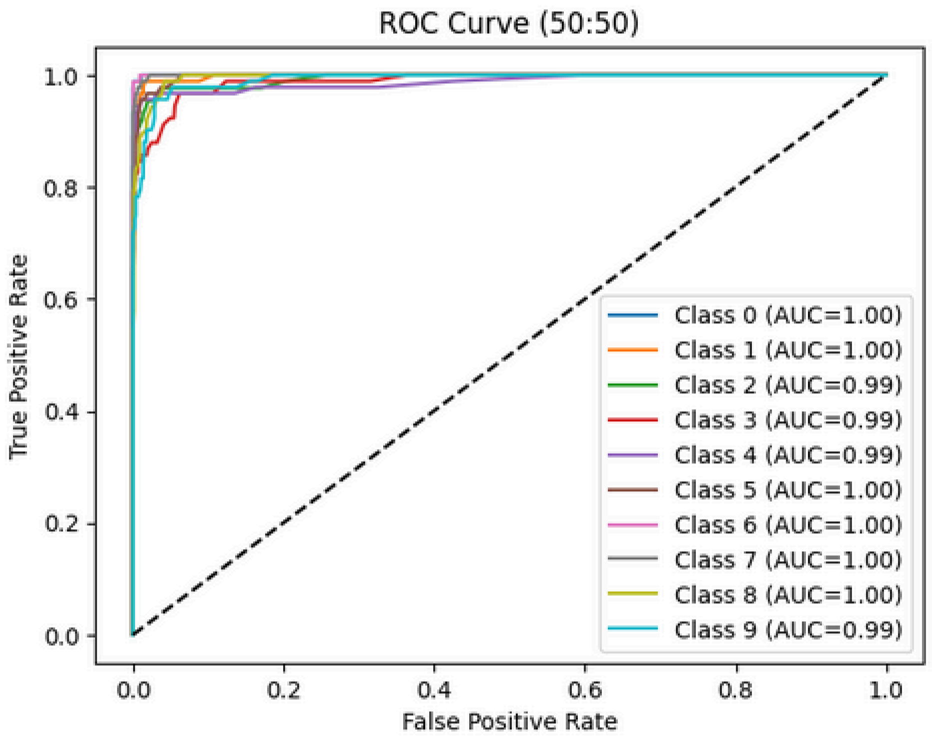
Train Test Split (50:50)

```
=== Train-Test Split: 50:50 ===
Best Parameters: {'criterion': 'entropy'}
              precision    recall  f1-score   support

           0       0.98      0.99      0.98        88
           1       0.93      0.92      0.93        91
           2       0.97      0.91      0.94        86
           3       0.92      0.86      0.89        91
           4       0.98      0.92      0.95        92
           5       0.88      0.91      0.90        91
           6       0.97      1.00      0.98        91
           7       0.95      0.97      0.96        89
           8       0.89      0.86      0.88        88
           9       0.85      0.96      0.90        92

    accuracy                           0.93       899
   macro avg       0.93      0.93      0.93       899
weighted avg       0.93      0.93      0.93       899
```



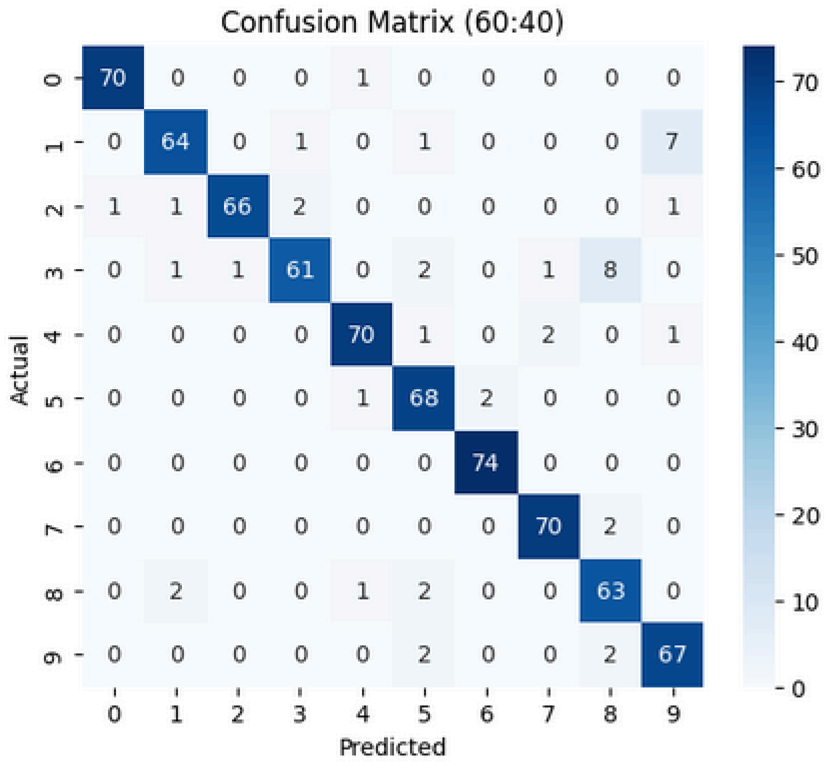
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	87	0	0	0	1	0	0	0	0	0
1	0	84	0	0	0	1	0	0	0	6
2	1	0	78	6	0	0	0	0	0	1
3	1	1	1	78	0	3	0	2	5	0
4	0	2	0	0	85	1	1	1	0	2
5	0	0	0	0	0	83	2	0	0	6
6	0	0	0	0	0	0	91	0	0	0
7	0	0	0	0	0	0	0	86	3	0
8	0	3	1	1	1	4	0	1	76	1
9	0	0	0	0	0	2	0	1	1	88



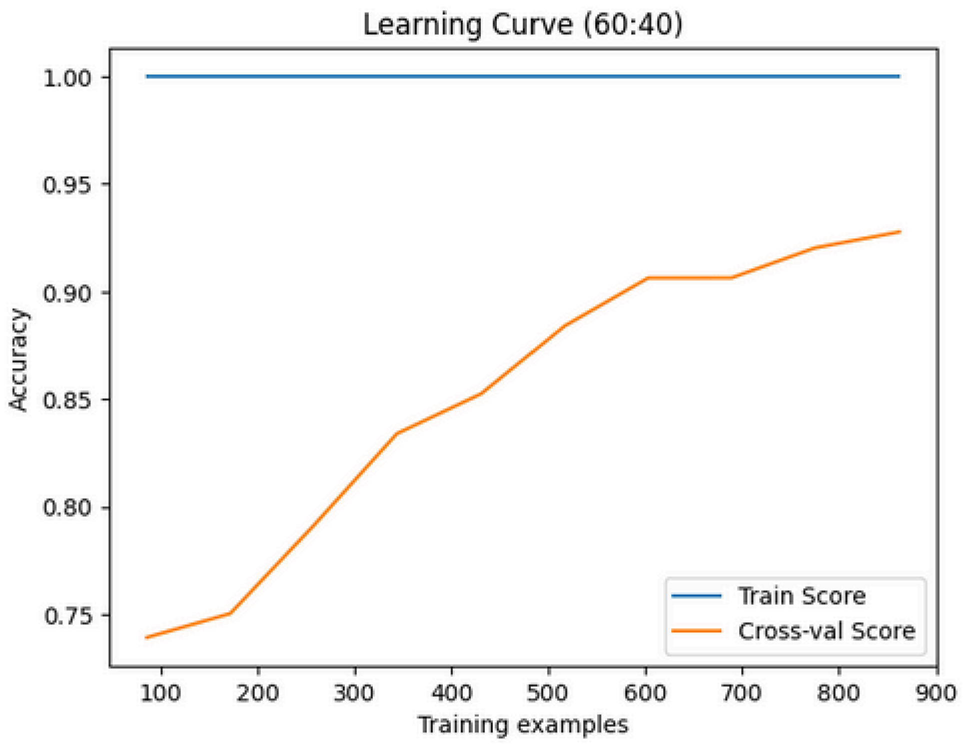


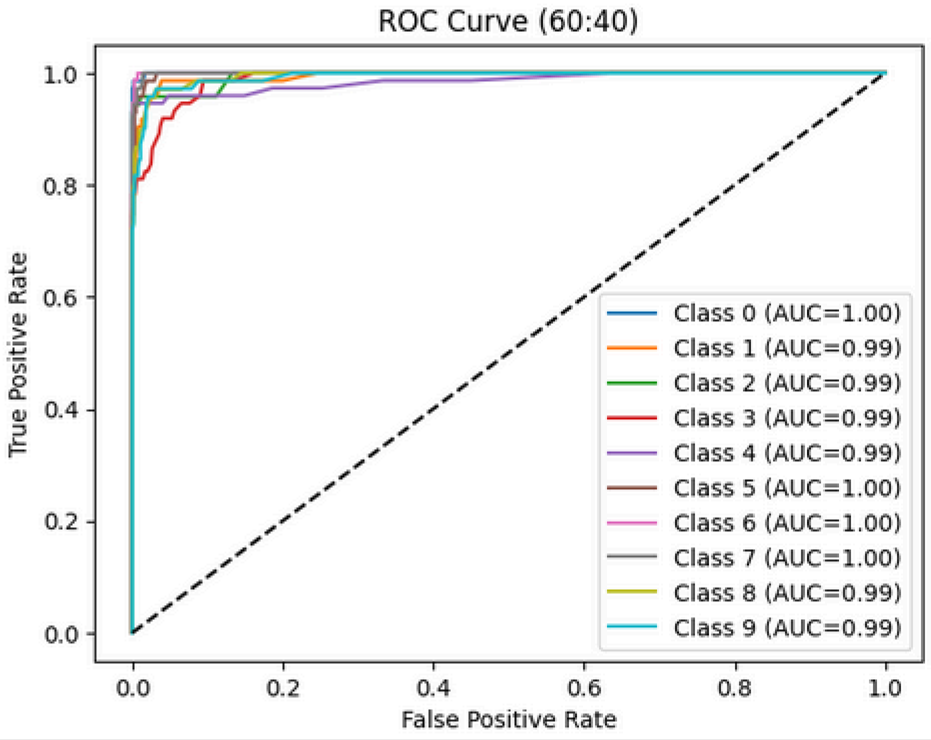
Train Test Split (60:40)

```
=== Train-Test Split: 60:40 ===
Best Parameters: {'criterion': 'entropy'}
              precision    recall  f1-score   support

           0       0.99      0.99      0.99        71
           1       0.94      0.88      0.91        73
           2       0.99      0.93      0.96        71
           3       0.95      0.82      0.88        74
           4       0.96      0.95      0.95        74
           5       0.89      0.96      0.93        71
           6       0.97      1.00      0.99        74
           7       0.96      0.97      0.97        72
           8       0.84      0.93      0.88        68
           9       0.88      0.94      0.91        71

    accuracy                           0.94       719
   macro avg       0.94      0.94      0.94       719
weighted avg       0.94      0.94      0.94       719
```



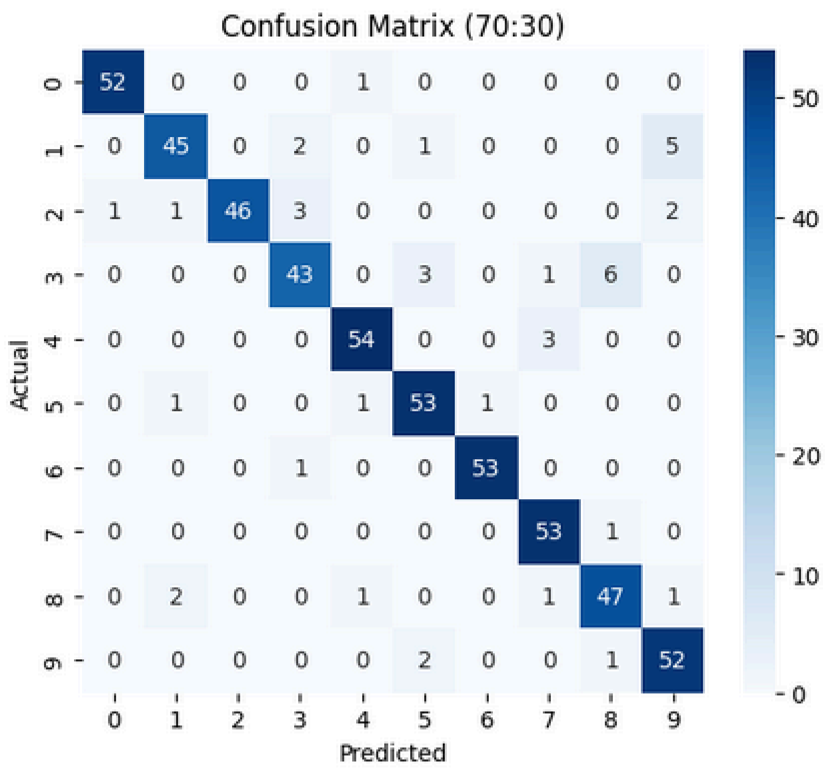
	0	1	2	3	4	5	6	7	8	9
0	70	0	0	0	1	0	0	0	0	0
1	0	64	0	1	0	1	0	0	0	7
2	1	1	66	2	0	0	0	0	0	1
3	0	1	1	61	0	2	0	1	8	0
4	0	0	0	0	70	1	0	2	0	1
5	0	0	0	0	1	68	2	0	0	0
6	0	0	0	0	0	0	74	0	0	0
7	0	0	0	0	0	0	0	70	2	0
8	0	2	0	0	1	2	0	0	63	0
9	0	0	0	0	0	2	0	0	2	67

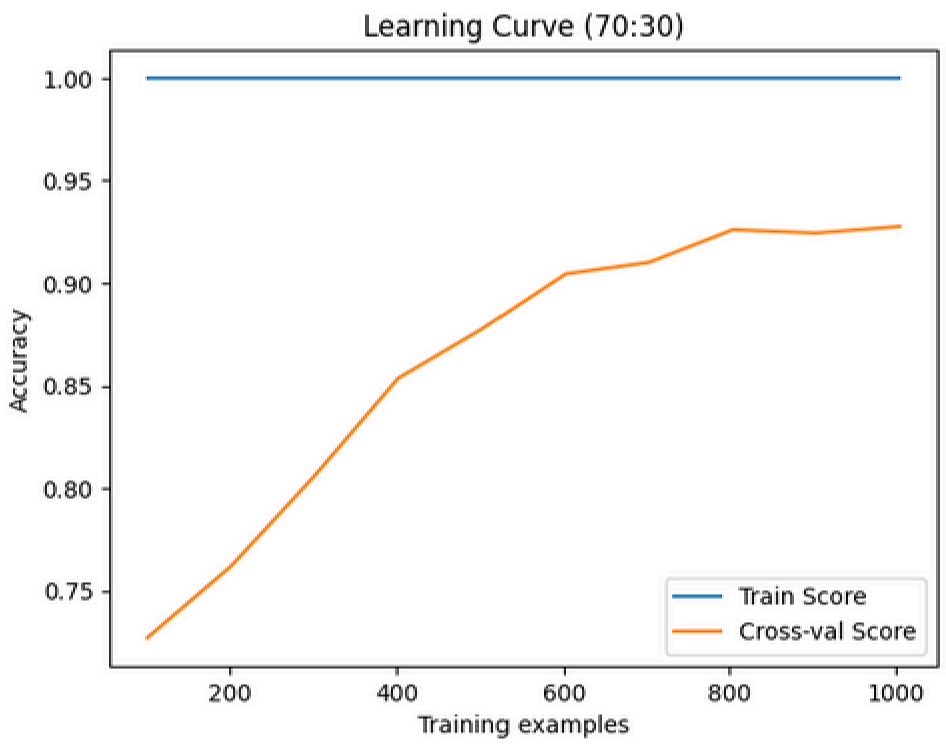


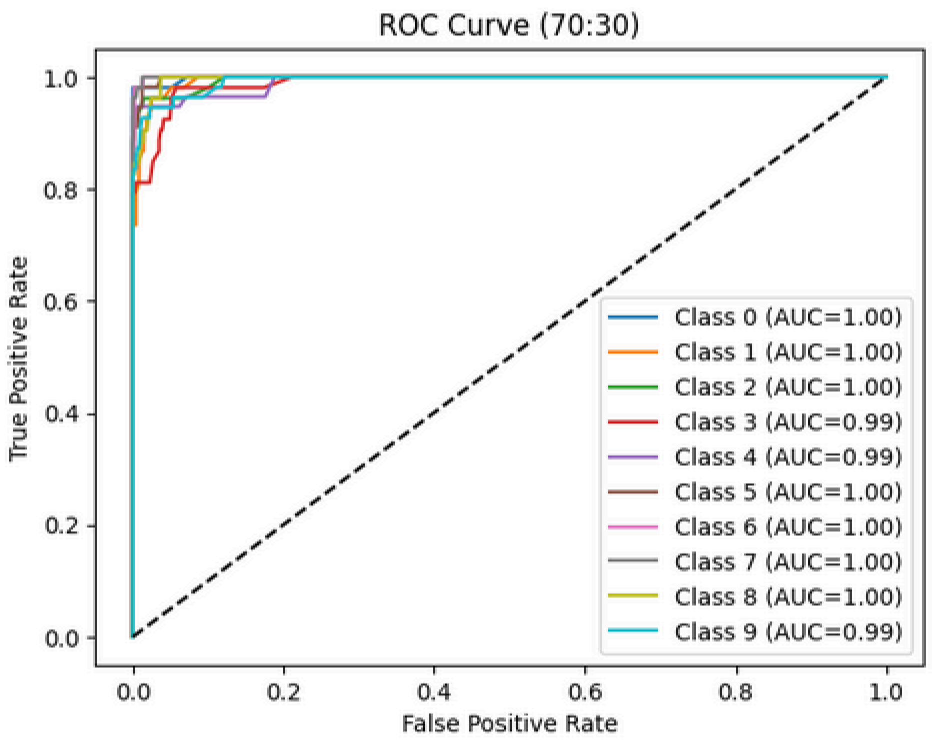


Train Test Split (70:30)

```
=== Train-Test Split: 70:30 ===
Best Parameters: {'criterion': 'entropy'}
              precision    recall  f1-score   support

           0       0.98      0.98      0.98        53
           1       0.92      0.85      0.88        53
           2       1.00      0.87      0.93        53
           3       0.88      0.81      0.84        53
           4       0.95      0.95      0.95        57
           5       0.90      0.95      0.92        56
           6       0.98      0.98      0.98        54
           7       0.91      0.98      0.95        54
           8       0.85      0.90      0.88        52
           9       0.87      0.95      0.90        55

    accuracy                           0.92       540
   macro avg       0.92      0.92      0.92       540
weighted avg       0.92      0.92      0.92       540
```

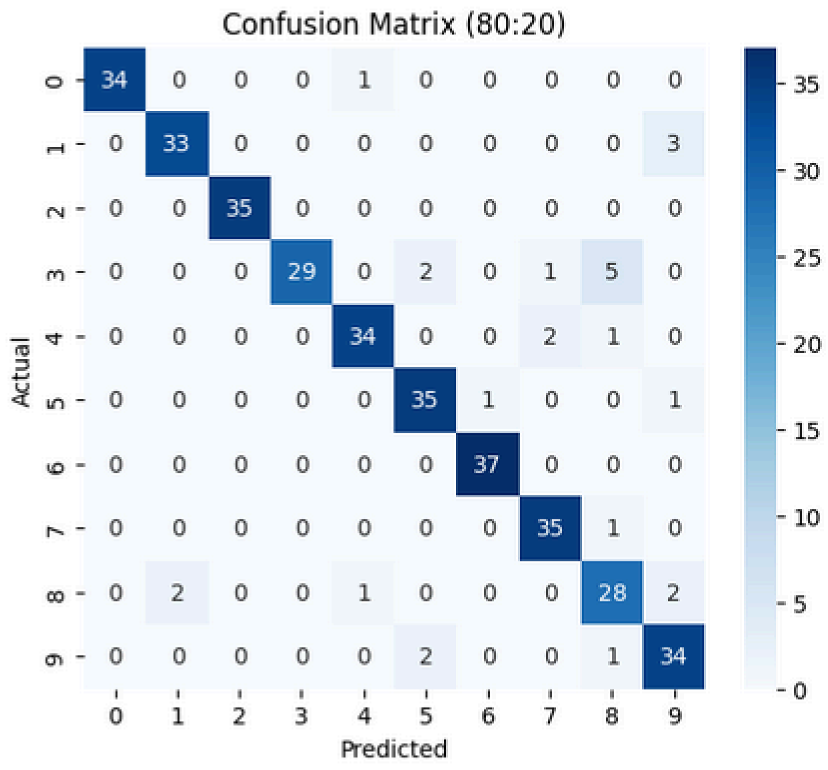




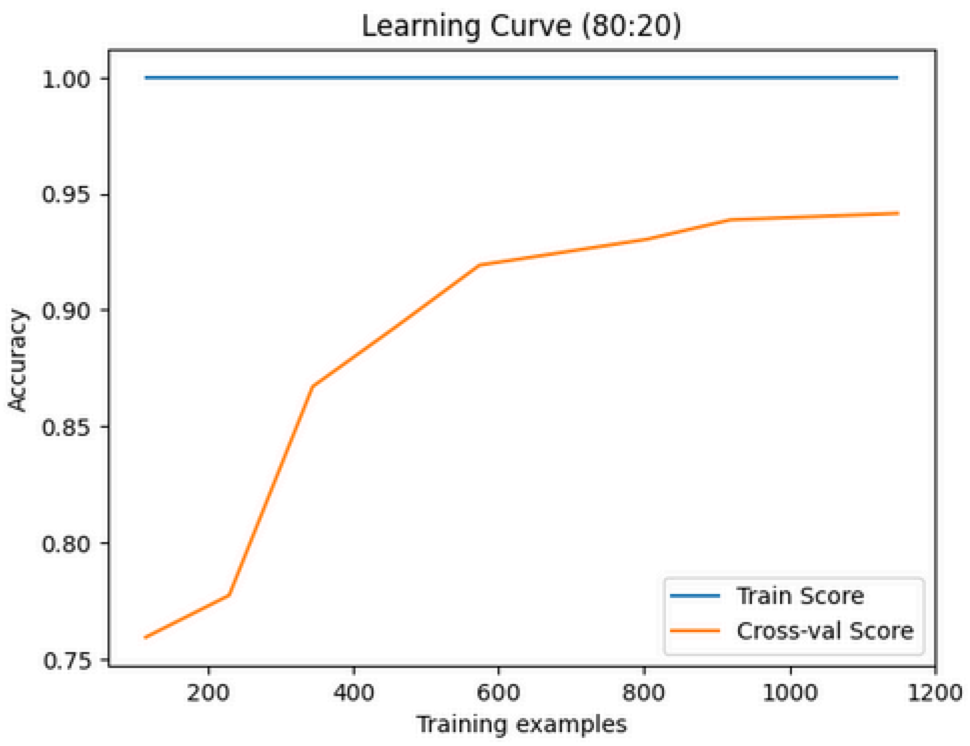


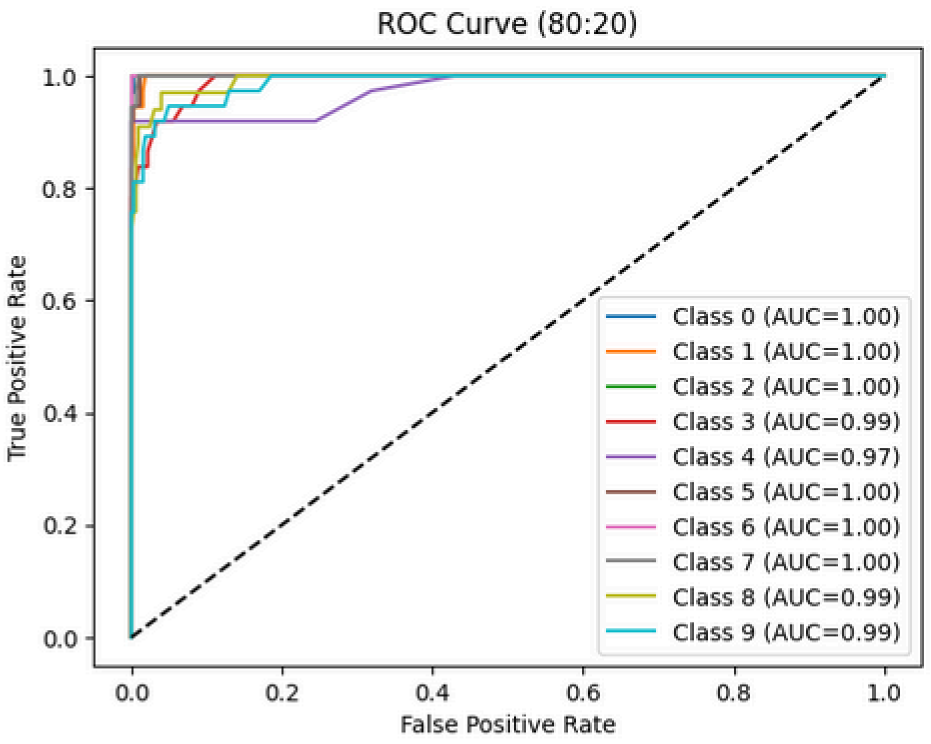
Train Test Split (80:20)

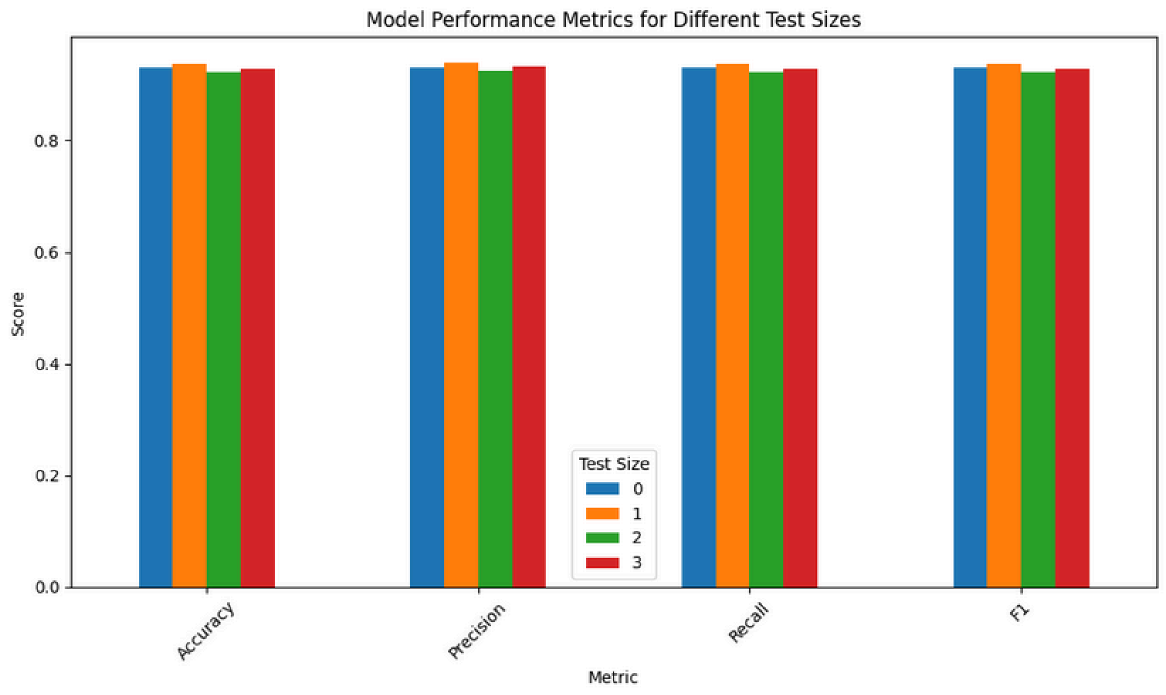
```
=== Train-Test Split: 80:20 ===
Best Parameters: {'criterion': 'entropy'}
              precision    recall  f1-score   support

           0       1.00      0.97      0.99        35
           1       0.94      0.92      0.93        36
           2       1.00      1.00      1.00        35
           3       1.00      0.78      0.88        37
           4       0.94      0.92      0.93        37
           5       0.90      0.95      0.92        37
           6       0.97      1.00      0.99        37
           7       0.92      0.97      0.95        36
           8       0.78      0.85      0.81        33
           9       0.85      0.92      0.88        37

    accuracy                           0.93       360
   macro avg       0.93      0.93      0.93       360
weighted avg       0.93      0.93      0.93       360
```



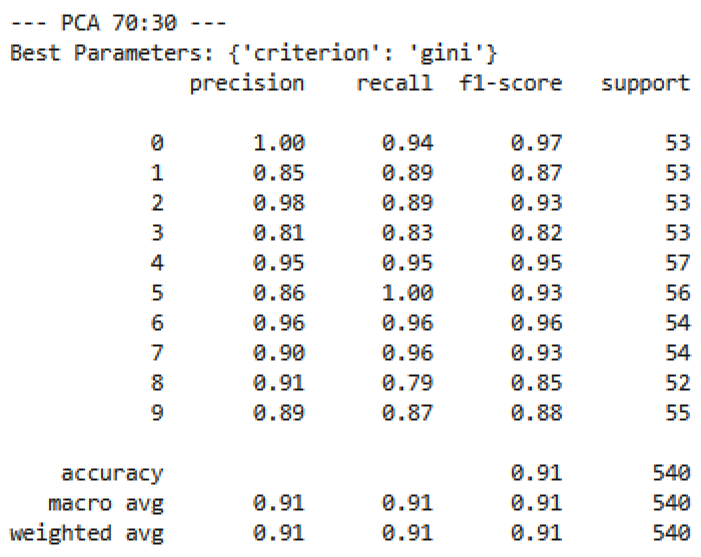
Actual \ Predicted	0	1	2	3	4	5	6	7	8	9
0	34	0	0	0	1	0	0	0	0	0
1	0	33	0	0	0	0	0	0	0	3
2	0	0	35	0	0	0	0	0	0	0
3	0	0	0	29	0	2	0	1	5	0
4	0	0	0	0	34	0	0	2	1	0
5	0	0	0	0	0	35	1	0	0	1
6	0	0	0	0	0	0	37	0	0	0
7	0	0	0	0	0	0	0	35	1	0
8	0	2	0	0	1	0	0	0	28	2
9	0	0	0	0	0	2	0	0	1	34


Learning Curve (80:20)
Accuracy
1.00
0.95
0.90
0.85
0.80
0.75
200
400
600
800
1000
1200
Training examples
Train Score
Cross-val Score

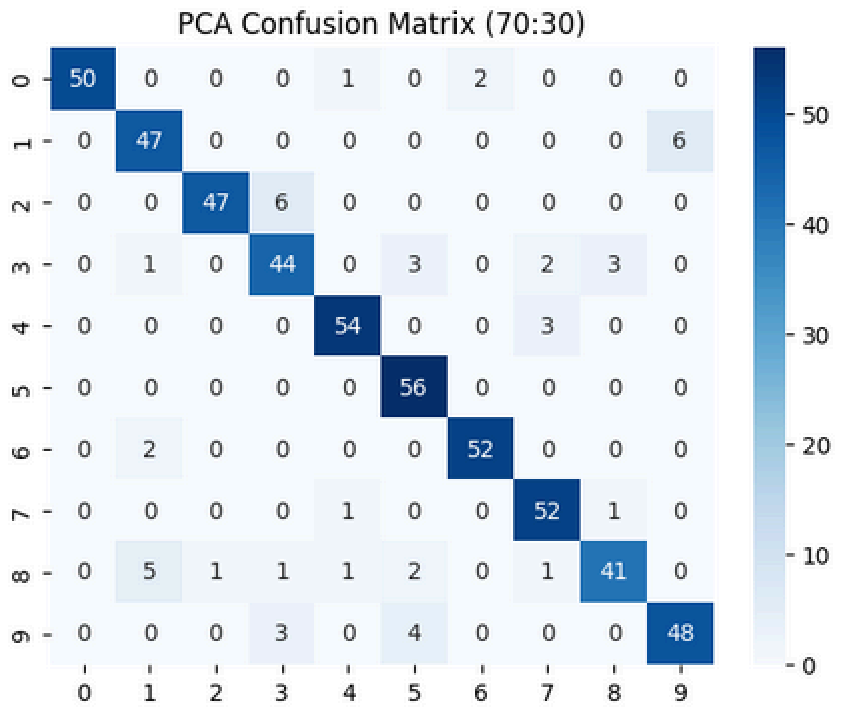

ROC Curve (80:20)
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
False Positive Rate
Class 0 (AUC=1.00)
Class 1 (AUC=1.00)
Class 2 (AUC=1.00)
Class 3 (AUC=0.99)
Class 4 (AUC=0.97)
Class 5 (AUC=1.00)
Class 6 (AUC=1.00)
Class 7 (AUC=1.00)
Class 8 (AUC=0.99)
Class 9 (AUC=0.99)



<u>Principal Component Analysis (PCA) for feature dimensionality reduction</u>



```
--- PCA 70:30 ---
Best Parameters: {'criterion': 'gini'}
              precision    recall  f1-score   support

           0       1.00      0.94      0.97        53
           1       0.85      0.89      0.87        53
           2       0.98      0.89      0.93        53
           3       0.81      0.83      0.82        53
           4       0.95      0.95      0.95        57
           5       0.86      1.00      0.93        56
           6       0.96      0.96      0.96        54
           7       0.90      0.96      0.93        54
           8       0.91      0.79      0.85        52
           9       0.89      0.87      0.88        55

    accuracy                           0.91       540
   macro avg       0.91      0.91      0.91       540
weighted avg       0.91      0.91      0.91       540
```



Discussion:

The comparative analysis of SVM, MLP, and Random Forest classifiers across the Wine and Digits datasets highlights distinct strengths and trade-offs of each method.

- SVM consistently delivered strong performance, particularly with the RBF kernel, which balanced bias and variance well. On the Wine dataset, SVM achieved high classification accuracy and robust F1-scores across different train-test splits. Similarly, for the Digits dataset, SVM with appropriate kernels (mainly RBF and polynomial) provided excellent generalization, making it a reliable baseline model. However, SVM training times increased with dataset size and parameter tuning.

- MLP (Neural Network) exhibited flexibility in learning complex decision boundaries. With careful hyperparameter tuning (hidden layers, activation, solver, and learning rate), MLP matched or exceeded SVM performance in some cases, especially on the Digits dataset where non-linear structures were more pronounced. Despite its adaptability, MLP was more sensitive to parameter choices and sometimes required longer training to converge.

- Random Forest offered robust and interpretable results with relatively less hyperparameter tuning effort. On both datasets, RF showed competitive accuracy and stable performance across splits, with lower variance compared to MLP. It was less computationally expensive than SVM with complex kernels, while still maintaining strong generalization. However, in high-dimensional representations, RF performance occasionally plateaued compared to tuned SVM/MLP models.

Overall, SVM emerged as the most consistent high-performer, particularly with kernel optimization, while MLP showed the highest potential when properly tuned. Random Forest provided the most stable and efficient performance with fewer tuning requirements, making it suitable when interpretability and lower training cost are priorities.
In conclusion, the choice of classifier depends on context: SVM is well-suited for precision-oriented tasks, MLP for capturing complex non-linear relationships, and Random Forest for practical, balanced performance.