MEX #4 - Geyzson Kristoffer

SN:2023-21036

https://uvle.upd.edu.ph/mod/assign/view.php?id=542086

In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.pipeline import make_pipeline
```

Problem a

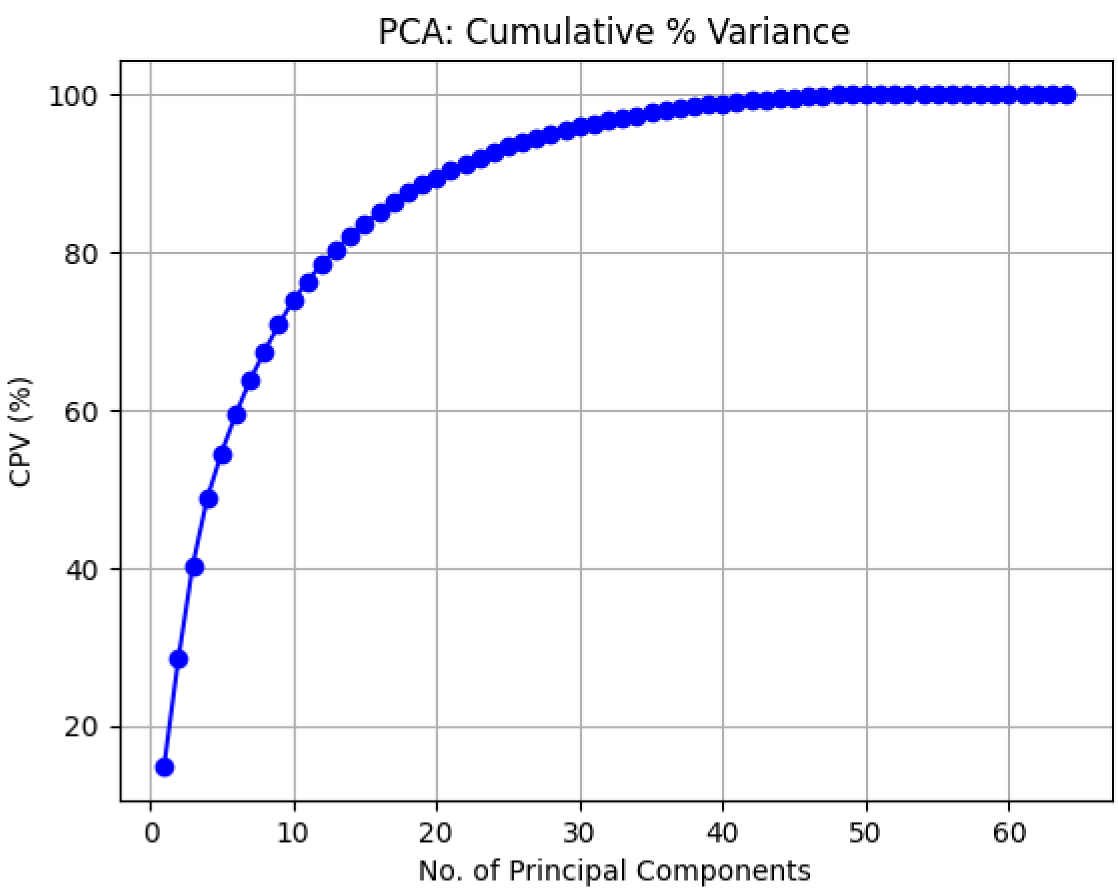
In [ ]:
```
X, y = load_digits(return_X_y=True)
target_names = load_digits().target_names

pca = PCA(n_components=2)
X_pca = pca.fit_transform(X, y)
var_digits = pca.explained_variance_ratio_
cpv = np.cumsum(var_digits)*100
print(f"Explained variance ratio:\t {var_digits}")
print(f"Cumulative percent variance:\t {cpv}")
```

```
Explained variance ratio:        [0.14890594 0.13618771]
Cumulative percent variance:     [14.89059358 28.50936482]
```

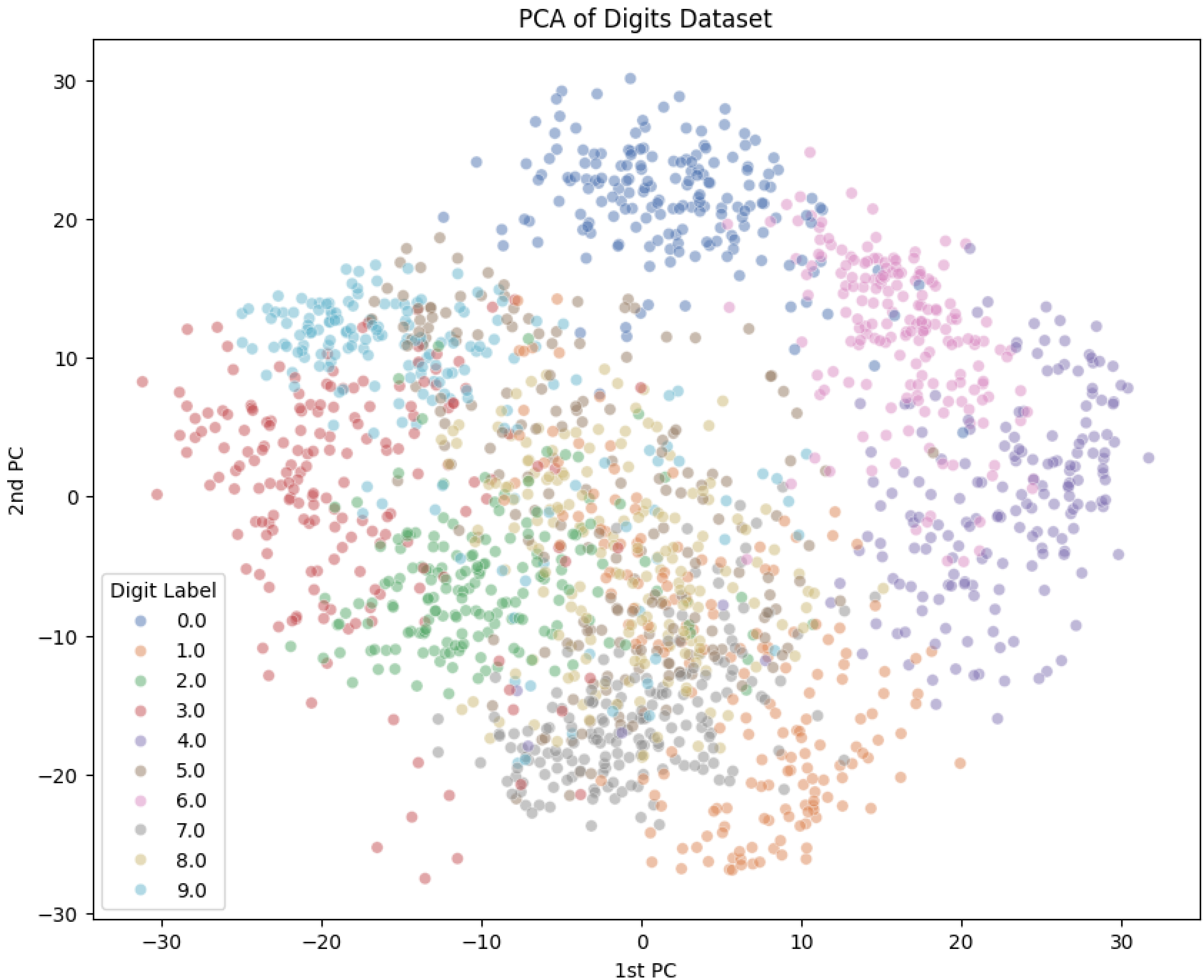
Problem a: What is the CPV at 2 PCs?

The Cumulative percent variance at 2 Principal Components is 28.51%

```
In [ ]: pca_full = PCA()
        X_pca_full = pca_full.fit_transform(X, y)
        var_digits_full = pca_full.explained_variance_ratio_
        cpv_full = np.cumsum(var_digits_full)*100
        plt.plot(np.arange(cpv_full.size)+1,cpv_full,'bo-')
        plt.title('PCA: Cumulative % Variance')
        plt.xlabel('No. of Principal Components')
        plt.ylabel('CPV (%)')
        plt.grid()
        plt.show()
```




```
In [ ]: df_digits = pd.DataFrame(data=np.c_[X_pca, y],
                                 columns=['1st PC', '2nd PC', 'Digit Label'])

        plt.figure(figsize=(10, 8))
        sns.scatterplot(data=df_digits, x="1st PC", y="2nd PC", hue="Digit Label", palette="deep", alpha=0.5)
        plt.title('PCA of Digits Dataset')
        plt.show()
```


PCA of Digits Dataset
2nd PC
1st PC
Digit Label
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0

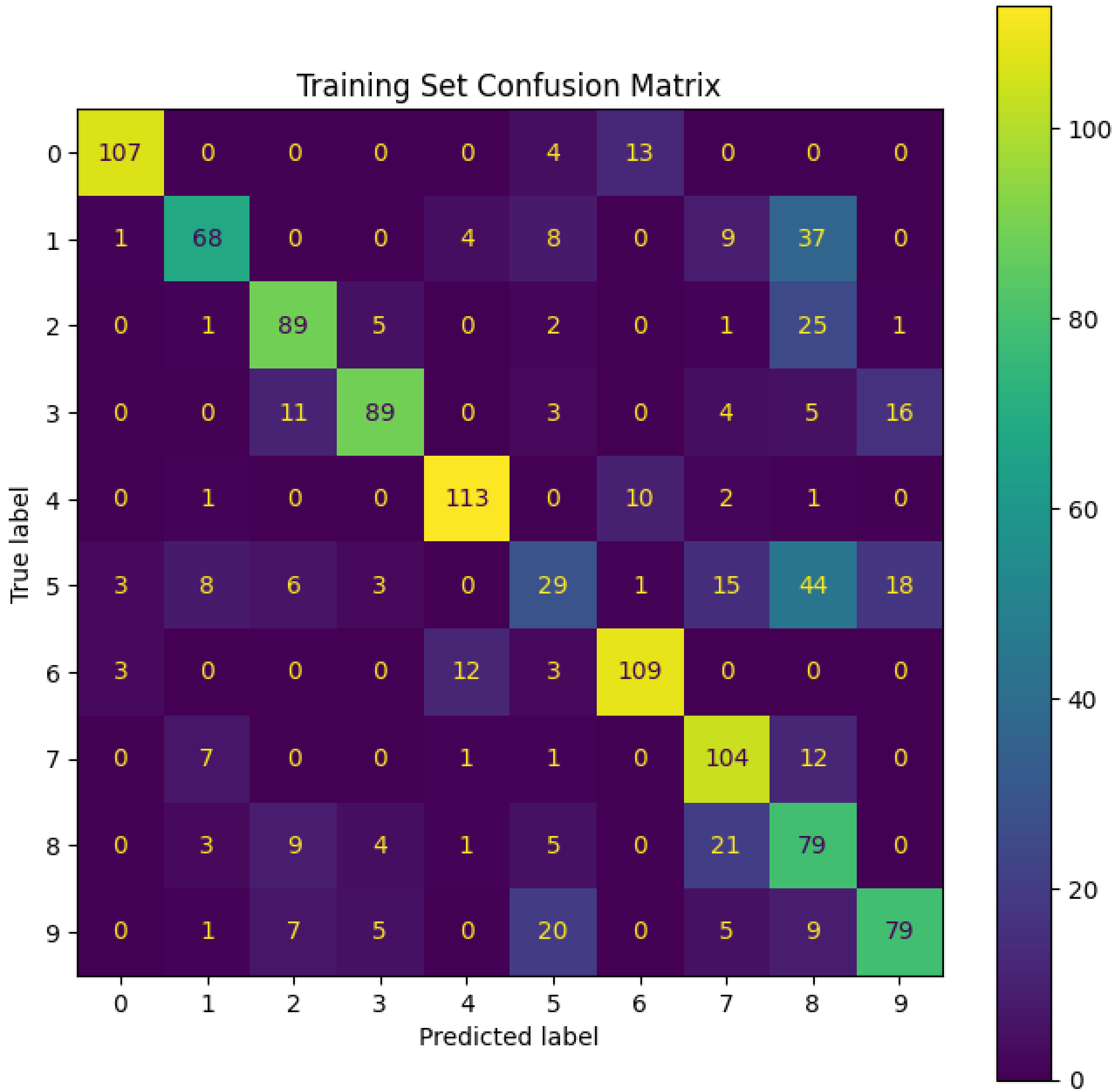
Problem b

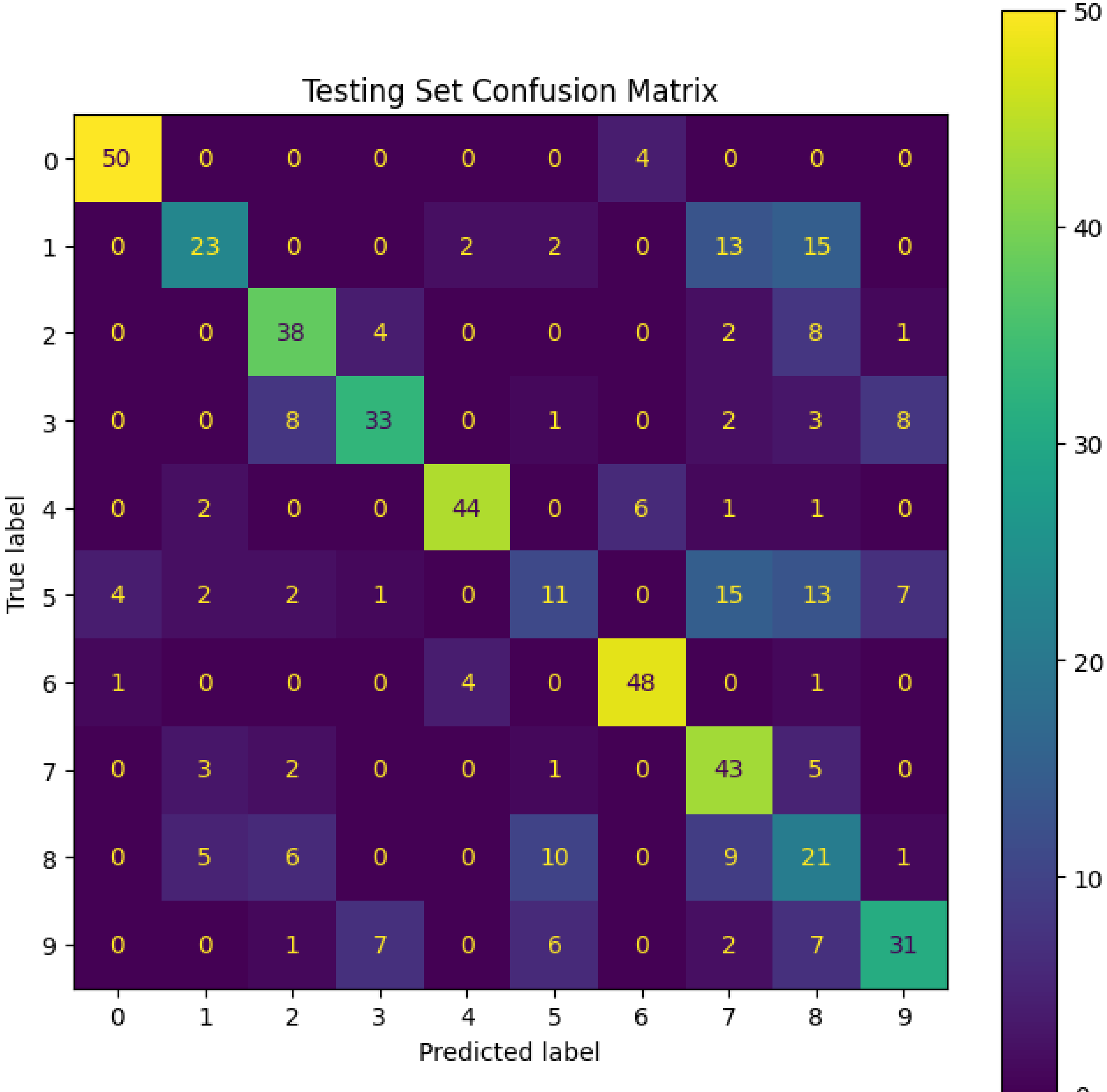
In [ ]:

```python
X_train, X_test, y_train, y_test = train_test_split(
    X_pca, y, test_size=0.3, stratify=y, random_state=0
)

param_grid = {
    'svc__C': [0.1, 1, 10, 100],
    'svc__gamma': ['scale', 'auto'],
    'svc__kernel': ['linear', 'rbf', 'poly'],
    'svc__degree': [2, 3, 4],
}

# param_grid = {
#     'svc__C': [0.1, 1, 10, 50, 100, 500, 1000],
#     'svc__gamma': [1e-3, 1e-2, 'scale', 'auto', 0.1, 0.5, 1],
#     'svc__kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
#     'svc__degree': [2, 3, 4],
#     'svc__class_weight': [None, 'balanced']
# }

model = make_pipeline(StandardScaler(), SVC())
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')

grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best parameters: {best_params}")
print(f"Best cross-validation accuracy: {best_score}")

best_svm = grid_search.best_estimator_

y_train_pred = best_svm.predict(X_train)
train_accuracy = best_svm.score(X_train, y_train)
train_conf_matrix = confusion_matrix(y_train, y_train_pred)

y_test_pred = best_svm.predict(X_test)
test_accuracy = best_svm.score(X_test, y_test)
test_conf_matrix = confusion_matrix(y_test, y_test_pred)

fig, ax = plt.subplots(figsize=(8, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=train_conf_matrix, display_labels=target_names)
disp.plot(ax=ax)
ax.set_title('Training Set Confusion Matrix')
print(f"Training Classification Accuracy: {train_accuracy}")

fig, ax = plt.subplots(figsize=(8, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=test_conf_matrix, display_labels=target_names)
disp.plot(ax=ax)
```

```
ax.set_title('Testing Set Confusion Matrix')
print(f"Testing Classification Accuracy: {test_accuracy}")
```

```
Best parameters: {'svc__C': 10, 'svc__degree': 2, 'svc__gamma': 'scale', 'svc__kernel': 'rbf'}
Best cross-validation accuracy: 0.6770157465376588
Training Classification Accuracy: 0.6889419252187748
Testing Classification Accuracy: 0.6333333333333333
```





Problem c

```
In [ ]: lda = LinearDiscriminantAnalysis(n_components=2)
        X_lda = lda.fit_transform(X, y)

        X_train_lda, X_test_lda, y_train_lda, y_test_lda = train_test_split(
```

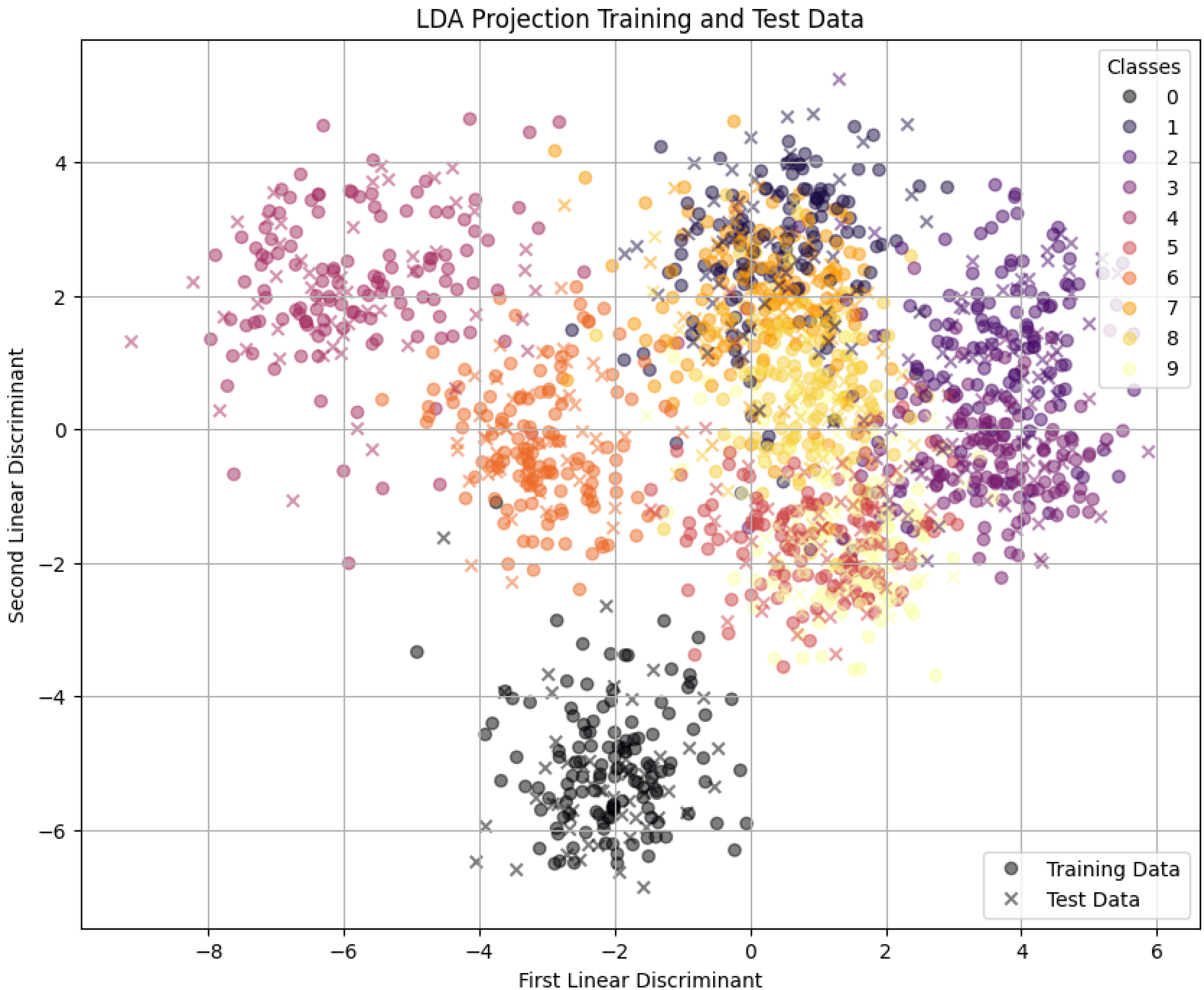
```
    X_lda, y, test_size=0.3, stratify=y, random_state=0
)

lda_classifier = LinearDiscriminantAnalysis()
lda_classifier.fit(X_train_lda, y_train_lda)
```

Out[ ]:

```
▾ LinearDiscriminantAnalysis
LinearDiscriminantAnalysis()
```

In [ ]:

```
plt.figure(figsize=(10, 8))

scatter1 = plt.scatter(X_train_lda[:, 0], X_train_lda[:, 1], c=y_train_lda,
                       cmap='inferno', alpha=0.5, marker='o', label='Training Data')

scatter2 = plt.scatter(X_test_lda[:, 0], X_test_lda[:, 1], c=y_test_lda,
                       cmap='inferno', alpha=0.5, marker='x', label='Test Data')

legend1 = plt.legend(*scatter1.legend_elements(), loc="upper right", title="Classes")
plt.gca().add_artist(legend1)

plt.legend(handles=[scatter1.legend_elements()[0][0], scatter2.legend_elements()[0][0]],
           labels=['Training Data', 'Test Data'], loc="lower right")

plt.title('LDA Projection Training and Test Data')
plt.xlabel('First Linear Discriminant')
plt.ylabel('Second Linear Discriminant')
plt.grid(True)
plt.show()
```

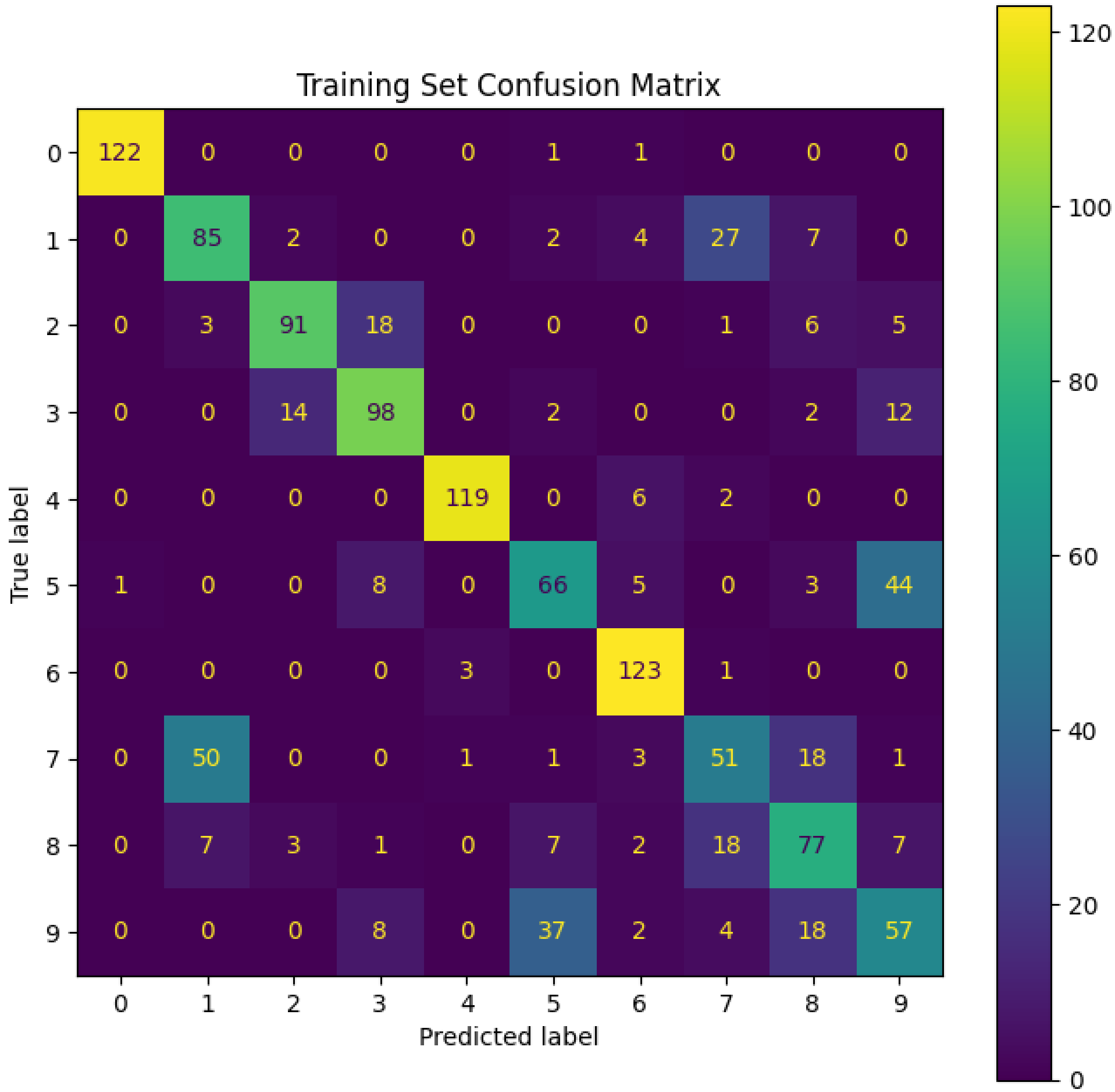


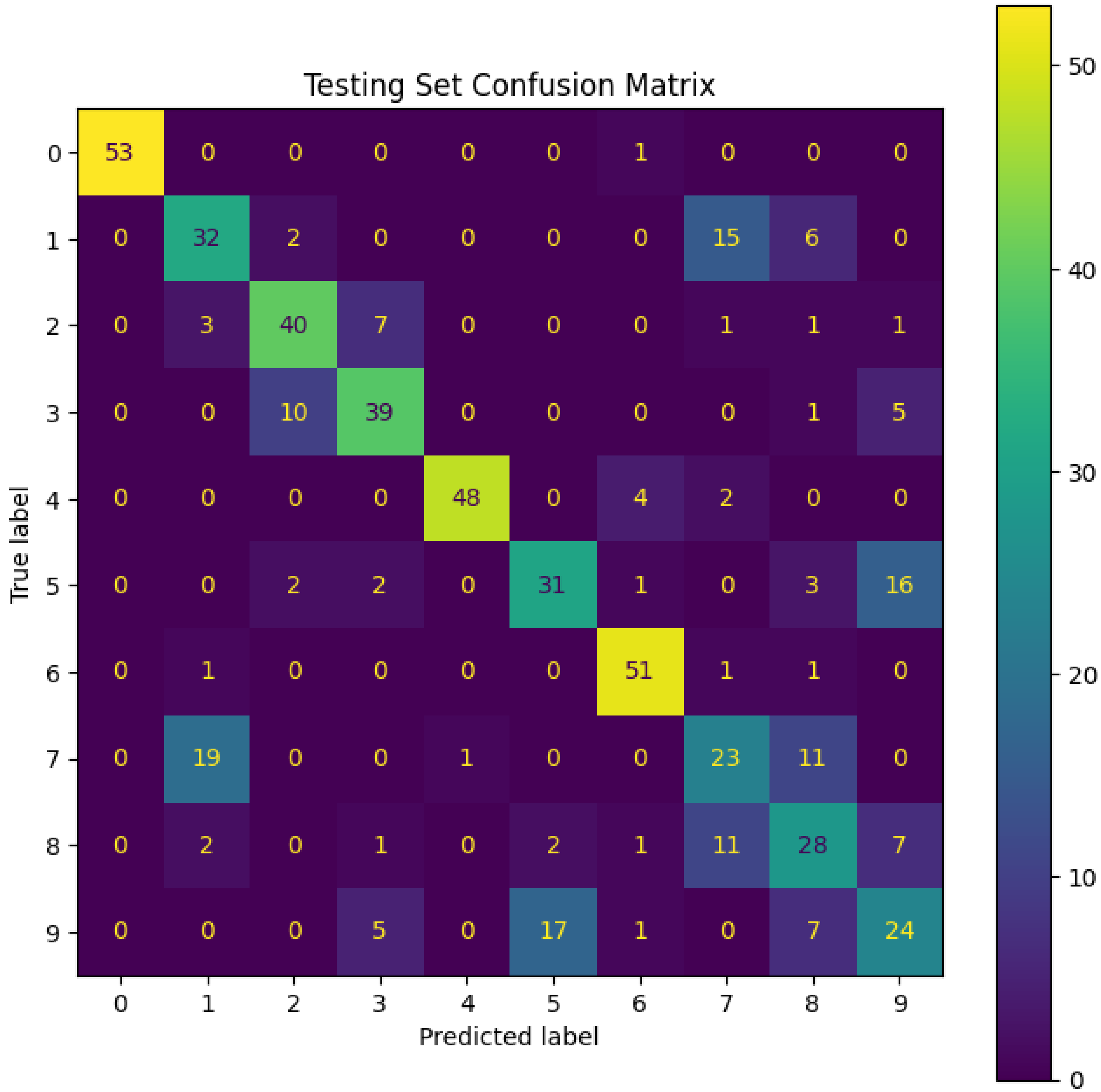
In [ ]:

```
y_train_pred_lda = lda_classifier.predict(X_train_lda)
train_accuracy_lda = lda_classifier.score(X_train_lda, y_train_lda)
train_conf_matrix_lda = confusion_matrix(y_train_lda, y_train_pred_lda)

y_test_pred_lda = lda_classifier.predict(X_test_lda)
```

```python
test_accuracy_lda = lda_classifier.score(X_test_lda, y_test_lda)
test_conf_matrix_lda = confusion_matrix(y_test_lda, y_test_pred_lda)

fig, ax = plt.subplots(figsize=(8, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=train_conf_matrix_lda, display_labels=target_names)
disp.plot(ax=ax)
ax.set_title('Training Set Confusion Matrix')
print(f"Training Classification Accuracy: {train_accuracy_lda}")

fig, ax = plt.subplots(figsize=(8, 8))
disp = ConfusionMatrixDisplay(confusion_matrix=test_conf_matrix_lda, display_labels=target_names)
disp.plot(ax=ax)
ax.set_title('Testing Set Confusion Matrix')
print(f"Testing Classification Accuracy: {test_accuracy_lda}")
```

```
Training Classification Accuracy: 0.7072394590294352
Testing Classification Accuracy: 0.6833333333333333
```


Training Set Confusion Matrix
True label
Predicted label



Question: Based on your results, which of the two methods above (PCA+SVM vs. LDA) is more preferrable for the 8x8 Handwritten Digits classification? Use various aspects for comparison such as computational effort, human effort, model accuracy, interpretability, etc.

Answer:

Preference depends on the goal of the modeler and the underlying motivations, as well as the system being used. The reason is that computationally, PCA+SVM is heavier since it requires training another SVM, which is in itself a very computationally expensive model.

From a human effort perspective, LDA still has an advantage as it is easier to set up than PCA+SVM. In SVM, you need to search for the best parameters to achieve higher accuracy. For LDA, the parameter options are more limited. In terms of model accuracy, both are fairly similar, although you can achieve higher accuracy with PCA+SVM if you invest more time in the model.

Overall, if you are pressed for time, LDA would be preferred, but if you aim to create a more sophisticated model, then PCA+SVM is the way to go.