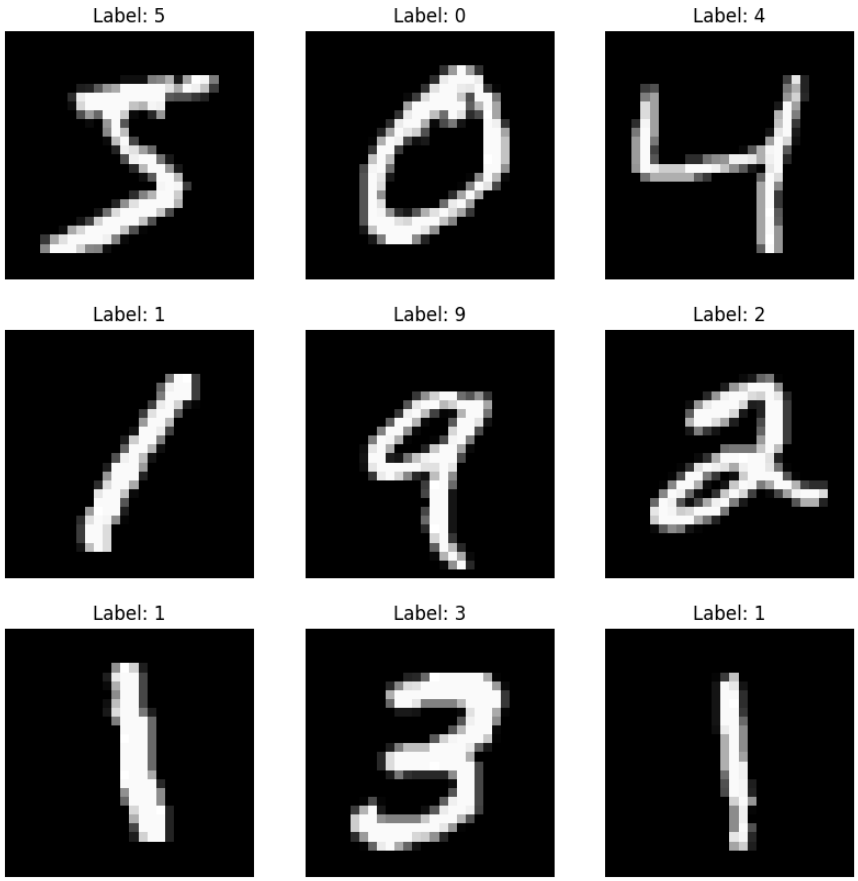
```
# Import basic libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# Import the libraries for performing classification
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import cross_val_score, train_test_split, GridSearchCV
from sklearn.metrics import classification_report, confusion_matrix


# Fetch the MNIST dataset from openml
from sklearn.datasets import fetch_openml

mnist = fetch_openml('mnist_784', version=1)
X, y = mnist["data"], mnist["target"]

# Convert target labels to integers
y = y.astype(np.int8)

# Check the shapes of the dataset
print("Shape of X (data):", X.shape)  # Should be (70000, 784)
print("Shape of y (labels):", y.shape)  # Should be (70000,)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` will change f
  warn(
Shape of X (data): (70000, 784)
Shape of y (labels): (70000,)
```

```
# Plot the first few images from the dataset
plt.figure(figsize=(10, 10))
for i in range(9):
    plt.subplot(3, 3, i+1)
    # Access the i-th row of the DataFrame using .iloc[]
    plt.imshow(X.iloc[i].values.reshape(28, 28), cmap='gray')
    plt.title(f"Label: {y[i]}")
    plt.axis('off')
plt.show()
```




```
# Flatten each input image into a vector of length 784 and normalize the data
X = X / 255.0  # Normalize

# Check the shape of the dataset
print("Shape after flattening and normalizing, X:", X.shape)
```

```
Shape after flattening and normalizing, X: (70000, 784)
```

```
# Splitting the dataset
X_train, X_test, y_train, y_test = X[:10000], X[10000:12000], y[:10000], y[10000:12000]

# Check the shape of the training and testing sets
print("Training data shape:", X_train.shape)
print("Testing data shape:", X_test.shape)
```

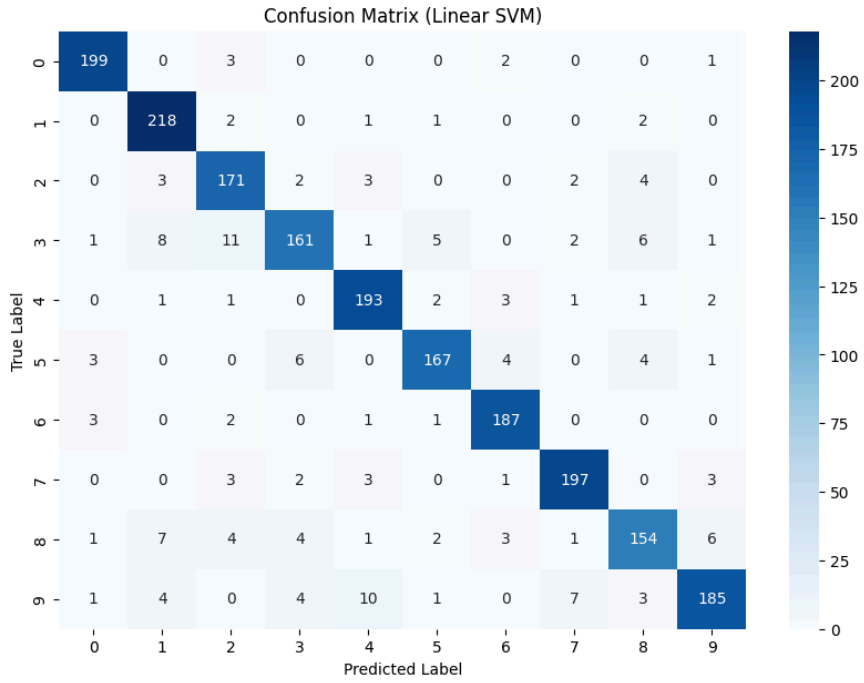
```
Training data shape: (10000, 784)
Testing data shape: (2000, 784)
```

```
# Train Linear SVM using a pipeline with MinMaxScaler and SVC with a linear kernel
pipe_1 = Pipeline([('scaler', MinMaxScaler()),
                   ('classifier', SVC(kernel='linear', C=1))])

# Fit the model
pipe_1.fit(X_train, y_train.ravel())

# Evaluate using cross-validation
acc = cross_val_score(pipe_1, X_train, y_train.ravel(), cv=2)
print("Training Accuracy with Linear SVM: {:.2f} %".format(acc.mean() * 100))
```

```
Training Accuracy with Linear SVM: 91.07 %
```

```
# Predicting on test data
y_pred = pipe_1.predict(X_test)
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Plotting heatmap for confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Linear SVM)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Classification report
print("Classification Report for Linear SVM:\n")
print(classification_report(y_test, y_pred))
```

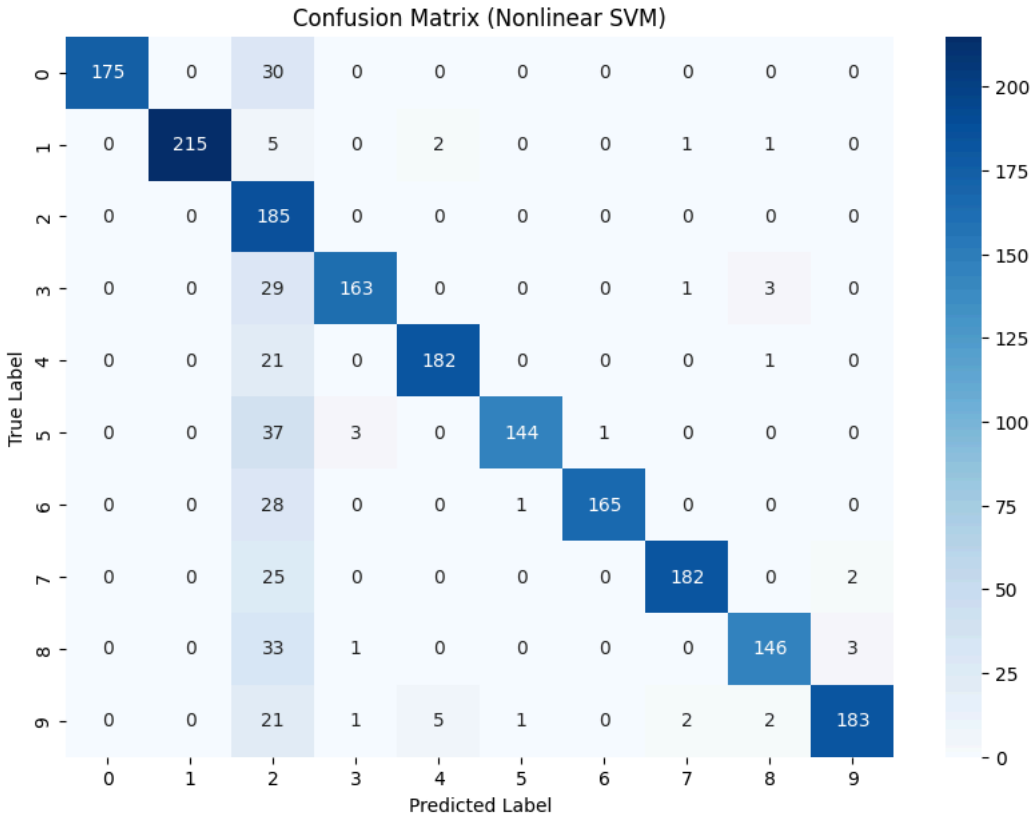



```
Classification Report for Linear SVM:

              precision    recall  f1-score   support

           0       0.96      0.97      0.96       205
           1       0.90      0.97      0.94       224
           2       0.87      0.92      0.90       185
           3       0.90      0.82      0.86       196
           4       0.91      0.95      0.93       204
           5       0.93      0.90      0.92       185
           6       0.94      0.96      0.95       194
           7       0.94      0.94      0.94       209
           8       0.89      0.84      0.86       183
           9       0.93      0.86      0.89       215

    accuracy                           0.92      2000
   macro avg       0.92      0.91      0.91      2000
weighted avg       0.92      0.92      0.92      2000
```

```
# Train Nonlinear SVM using a pipeline with MinMaxScaler and SVC with RBF kernel
pipe_2 = Pipeline([('scaler', MinMaxScaler()),
                   ('classifier', SVC(kernel='rbf', gamma=0.1, C=1))])

# Fit the model
pipe_2.fit(X_train, y_train.ravel())

# Evaluate using cross-validation
acc = cross_val_score(pipe_2, X_train, y_train.ravel(), cv=2)
print("Training Accuracy with Nonlinear SVM: {:.2f} %".format(acc.mean() * 100))
```

```
Training Accuracy with Nonlinear SVM: 82.87 %
```

```
# Predicting on test data
y_pred_rbf = pipe_2.predict(X_test)

# Confusion matrix
conf_matrix_rbf = confusion_matrix(y_test, y_pred_rbf)

# Plotting heatmap for confusion matrix
plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_rbf, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Nonlinear SVM)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Classification report
print("Classification Report for Nonlinear SVM:\n")
print(classification_report(y_test, y_pred_rbf))
```




```
Classification Report for Nonlinear SVM:

              precision    recall  f1-score   support

           0       1.00      0.85      0.92       205
           1       1.00      0.96      0.98       224
           2       0.45      1.00      0.62       185
           3       0.97      0.83      0.90       196
           4       0.96      0.89      0.93       204
           5       0.99      0.78      0.87       185
           6       0.99      0.85      0.92       194
           7       0.98      0.87      0.92       209
           8       0.95      0.80      0.87       183
           9       0.97      0.85      0.91       215

    accuracy                           0.87      2000
   macro avg       0.93      0.87      0.88      2000
weighted avg       0.93      0.87      0.89      2000
```

```
# Define parameter grid
param_grid = {
    'classifier__C': [0.1, 1, 10],
    'classifier__gamma': [0.01, 0.1, 1]
}

# Set up the pipeline
pipe = Pipeline([('scaler', MinMaxScaler()),
                 ('classifier', SVC(kernel='rbf'))])

# Use GridSearchCV to find the best parameters
grid_search = GridSearchCV(pipe, param_grid, cv=3, scoring='accuracy')
grid_search.fit(X_train, y_train)
```

```
# Best parameters
print("Best parameters found by GridSearch:", grid_search.best_params_)

# Evaluate best model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

# Confusion matrix and classification report for the best model
conf_matrix_best = confusion_matrix(y_test, y_pred_best)

plt.figure(figsize=(10, 7))
sns.heatmap(conf_matrix_best, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix (Best Model)")
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.show()

# Print classification report
print("Classification Report for Best Model:\n")
print(classification_report(y_test, y_pred_best))
```

```
Best parameters found by GridSearch: {'classifier__C': 10, 'classifier__gamma': 0.01}
```