Support Vector Machine (SVM) Classification Report for Hand-written Digits Dataset

Workflow and Code Implementation

Data Preparation

Step 1: Load Dataset

```
from sklearn.datasets import load_digits
import pandas as pd

# Load digits dataset
digits = load_digits()
X = pd.DataFrame(data=digits.data, columns=digits.feature_names)
Y = pd.DataFrame(digits.target, columns=['target'])
```

Step 2: Data Visualization

- Displayed class distribution:

```
import seaborn as sns
import matplotlib.pyplot as plt

sns.barplot(Y['target'].value_counts(), palette='rainbow')
plt.title('Class distribution')
plt.xlabel('Digit')
plt.ylabel('Frequency')
plt.show()
```

- Visualized sample images with corresponding labels:

```
fig, axes = plt.subplots(1, 10, figsize=(15, 3))
for i, ax in enumerate(axes):
    ax.imshow(X.iloc[i].to_numpy().reshape(8, 8), cmap='gray')
    ax.axis('off')
    ax.set_title(f"Label: {digits.target[i]}")
plt.show()
```

Step 3: Statistical Analysis

- Analyzed mean pixel intensities:

```
import numpy as np
mean_intensity = [np.mean(digits.images[digits.target == i]) for i in
range(10)]
sns.barplot(x=range(10), y=mean_intensity, palette='tab10')
plt.title("Mean Pixel Intensity by Digit")
plt.xlabel("Digit")
plt.ylabel("Mean Intensity")
plt.show()
```

- Displayed average heatmaps for each digit:

```
mean_images = [np.mean(digits.images[digits.target == i], axis=0) for i in
range(10)]
fig, axes = plt.subplots(1, 10, figsize=(15, 3))
for i, ax in enumerate(axes):
    ax.imshow(mean_images[i], cmap='hot')
    ax.axis('off')
    ax.set_title(f"Digit: {i}")
plt.show()
```

Step 4: Data Scaling

- Standardized the features:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

Step 5: Train-Test Split

- Split data for training and testing:

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y,
test_size=0.2, random_state=0, stratify=Y)
```

Model Training and Hyperparameter Tuning

Step 6: SVM Training with Grid Search

- Performed hyperparameter tuning:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

param_grid = {'kernel': ['linear', 'polynomial', 'rbf']}
svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5)
grid_search.fit(X_train, Y_train)
```

```
best_model = grid_search.best_estimator_
print(f"Best Parameters: {grid_search.best_params_}")
```

Step 7: Evaluation Metrics

- Computed performance metrics:

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score

Y_pred = best_model.predict(X_test)
accuracy = accuracy_score(Y_test, Y_pred)
precision = precision_score(Y_test, Y_pred, average='weighted')
recall = recall_score(Y_test, Y_pred, average='weighted')
f1 = f1_score(Y_test, Y_pred, average='weighted')

print(f"Accuracy: {accuracy:.2f}")
print(f"Precision: {precision:.2f}")
print(f"Recall: {recall:.2f}")
print(f"F1 Score: {f1:.2f}")
```

Misclassification Analysis

Step 8: Confusion Matrix

- Generated and visualized the confusion matrix:

```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(Y, best_model.predict(X))
plt.figure(figsize=(6, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=digits.target_names, yticklabels=digits.target_names)
plt.title("Confusion Matrix for SVM")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

Observations

- **Strong Performance**:
  - High diagonal values indicate correct classifications.
  - Most classes achieved over 95% accuracy.
- **Misclassifications**:
  - Digits which have similar shapes got confused by the model:
    - Digit 4 often misclassified as 1, 6 due to overlapping features.
    - Digit 7 and 1: from color intensity diagram, 1 and 8 have similar intensities
    - Digit 8 and 1

Insights and Conclusions

1. **Performance Overview**:
    - Accuracy: **98%**
    - Precision: **98%**
    - Recall: **98%**
    - F1 Score: **98%**
2. **Misclassification Insights**:
    - The model struggles with visually similar digits.
    - The model has some imbalance toward class 1, it could be solved by reducing the class weight.
    - Adding more training data or feature engineering could improve separability.
3. **Recommendations**:
    - Investigate additional preprocessing (e.g., feature selection, augmentation).
    - Explore alternative models e Random Forest or deep learning approaches (e.g., CNNs).