ML Lab

Assignment 2:

Machine Learning Classification on Wine and Digits Datasets:

Roll no: 002211001065
Name:Abhijit Das
Sec: A1

Github Link:
https://github.com/abhijitdas6371/ML_LAB/blob/main/Ass2/Assignment2_ML_Lab.ipynb

1. **Problem Statement**

The goal is to build and evaluate classification models on two datasets: the Wine dataset and the Digits dataset. The task is to accurately predict the class labels of samples based on their feature values.

a. **Wine dataset**: Predict the wine class (class_0, class_1, class_2) based on chemical analysis of wines grown in the same region in Italy.
b. **Digits dataset**: Predict the digit (0 through 9) from 8x8 pixel grayscale images of handwritten digits.

2. **Discussion on different algorithms**

Support Vector Machine (SVM):

A supervised machine learning algorithm used for classification and regression tasks. It works by finding the optimal hyperplane that separates data points of different classes with the maximum margin.
For non-linearly separable data, SVM uses kernel functions to project data into higher dimensions where a separating hyperplane can be found.

Key Features:

1. Effective in high-dimensional spaces.
2. Can handle both linear and non-linear classification.
3. Works well with clear margin of separation.
4. Sensitive to parameter tuning and scaling of features.

Parameter Explanation:

- C: Regularization parameter; smaller values → smoother boundary (less overfitting), larger values → stricter classification.
- kernel: Defines the function to transform input data:
  - 'linear' → linear separation
  - 'poly' → polynomial transformation
  - 'rbf' → radial basis function (Gaussian, common for non-linear data)
  - 'sigmoid' → logistic-like function
- degree: Degree of polynomial kernel (used when kernel='poly').
- gamma: Defines influence of single training example; low = far influence, high = close influence.
- probability: If True, enables probability estimates (slower).

Multi-Layer Perceptron (MLP / Neural Network)

A type of artificial neural network used for classification and regression. It consists of input, hidden, and output layers with interconnected neurons. Each neuron applies a weighted sum of inputs followed by an activation function to introduce non-linearity.

Key Features:

1. Can model complex non-linear relationships.
2. Supports multi-class classification.
3. Requires scaling of data for better performance.
4. Sensitive to hyperparameters (hidden layers, learning rate, etc.).

Parameter Explanation:

- hidden_layer_sizes: Tuple defining the number of neurons in each hidden layer (e.g., (100,) = 1 hidden layer with 100 neurons).
- alpha: L2 regularization parameter to reduce overfitting.
- learning_rate_init: Step size at each iteration while moving toward a minimum.
- max_iter: Maximum number of iterations for training.
- activation: Non-linear activation function:
  - 'relu' → default, fast convergence
  - 'tanh' → smooth outputs, can be slower
  - 'logistic' → sigmoid function
- solver: Weight optimization algorithm:

    - 'adam' → stochastic gradient-based (default)
    - 'sgd' → stochastic gradient descent
    - 'lbfgs' → quasi-Newton optimizer

Random Forest (RF):

An ensemble learning algorithm that builds multiple Decision Trees and combines their predictions. It uses bagging (bootstrap aggregating) to create diverse trees and averages results (for regression) or uses majority voting (for classification).

Key Features:

1. Reduces overfitting compared to a single decision tree.
2. Handles both categorical and numerical data.
3. Provides feature importance.
4. More computationally expensive than a single tree.

Parameter Explanation:

- n_estimators: Number of trees in the forest.
- max_depth: Maximum depth of each tree. Controls overfitting.
- min_samples_split: Minimum number of samples required to split a node.
- min_samples_leaf: Minimum number of samples at a leaf node.
- criterion: Function to measure split quality ('gini' or 'entropy').
- max_features: Number of features considered for each split:
    - 'sqrt' → square root of total features (default for classification)
    - 'log2' → logarithm base 2 of features
    - None → all features considered
- bootstrap: Whether bootstrap samples are used when building trees (default=True).

3. **Dataset_Download**

```python
# Load Wine dataset

wine = load_wine()
X = wine.data
y = wine.target
classes = wine.target_names
```

```python
print(X.shape)
print(y.shape)
```

```
(178, 13)
(178,)
```

SVM without PCA

4. **Function to train the model, compute accuracy, precision, recall, F1 score, plot confusion matrix and ROC curves if available**

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.svm import SVC
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc
)
def evaluate_model(model, X_train, X_test, y_train, y_test, model_name="Model"):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted", zero_division=0)
    rec = recall_score(y_test, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_test, y_pred, average="weighted", zero_division=0)
    print(f"\n{model_name} Results:")
    print(f"Accuracy={acc:.4f}, Precision={prec:.4f}, Recall={rec:.4f}, F1={f1:.4f}")
    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=classes, yticklabels=classes)
    plt.title(f"Confusion Matrix ({model_name})")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
    roc_data = None
```

```
    if hasattr(model, "predict_proba"):
        y_bin = label_binarize(y_test, classes=np.arange(len(classes)))
        y_score = model.predict_proba(X_test)

        fpr, tpr, roc_auc = {}, {}, {}
        for i in range(len(classes)):
            fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_score[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])
        roc_data = (fpr, tpr, roc_auc)

    return acc, prec, rec, f1, roc_data
```

Define SVM classifier and hyperparameter search space for grid tuning

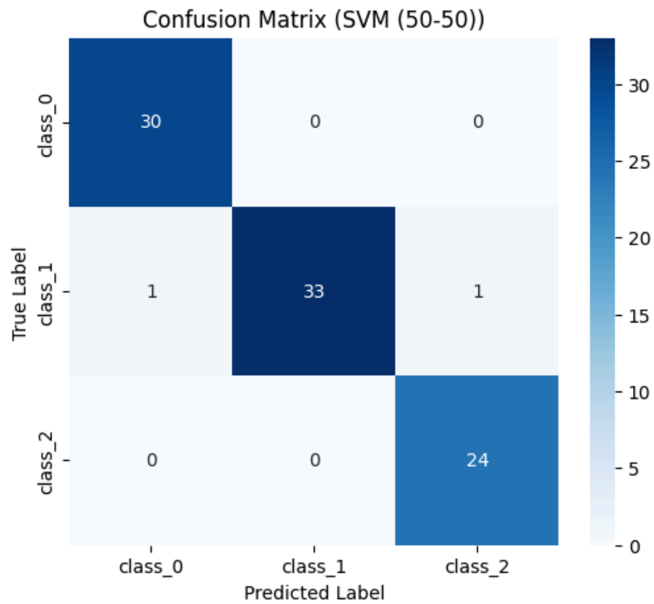
```
svm_model = SVC(probability=True, random_state=42)
svm_params = {
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "C": [0.1, 1, 10],
    "degree": [2, 3],
    "gamma": ["scale", "auto"]
}
```

5. Perform train-test splits, standardize data, tune SVM hyperparameters with HalvingGridSearchCV, evaluate metrics, and collect ROC data

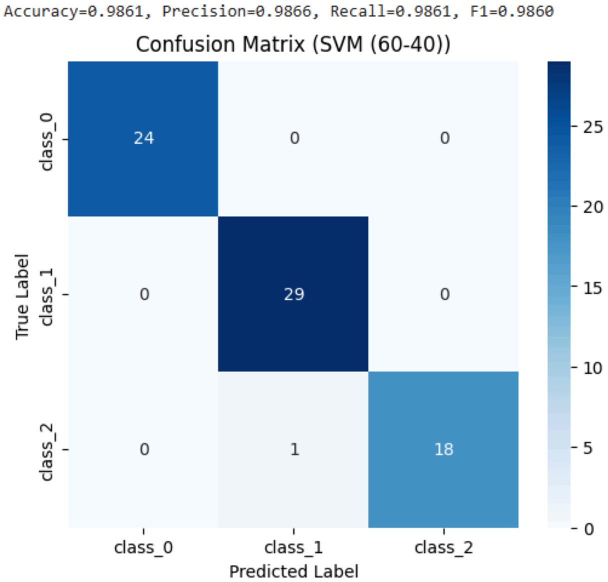
```
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary = []
roc_collector = {}

# Loop for SVM
for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label}\n{'='*50}")

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

    # Standardize features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Halving Grid Search
    grid = HalvingGridSearchCV(
        svm_model, svm_params, cv=5, scoring="accuracy",
        n_jobs=-1, random_state=42, verbose=0
    )
    grid.fit(X_train, y_train)

    results = pd.DataFrame(grid.cv_results_)
```

```python
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for SVM: {best_params}")

    best_model = SVC(**best_params, probability=True, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train, X_test, y_train, y_test,
        model_name=f"SVM ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary.append([split_label, acc, prec, rec, f1, params_str])


    if roc_data:
        roc_collector[split_label] = roc_data

else:
    print(f"No valid SVM models under 1.0 accuracy for split {split_label}")
```

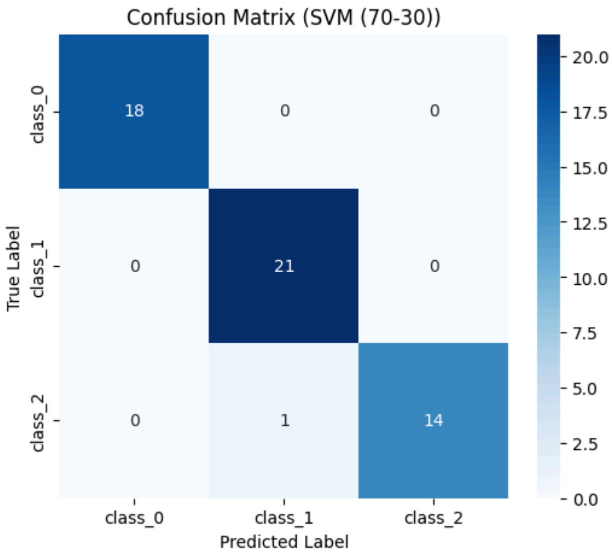
```
==================================================
 Train-Test Split 50-50
==================================================
Using best params for SVM: {'C': np.float64(0.1), 'degree': np.int64(2), 'gamma': 'scale', 'kernel': 'linear'}

SVM (50-50) Results:
Accuracy=0.9775, Precision=0.9783, Recall=0.9775, F1=0.9774
```



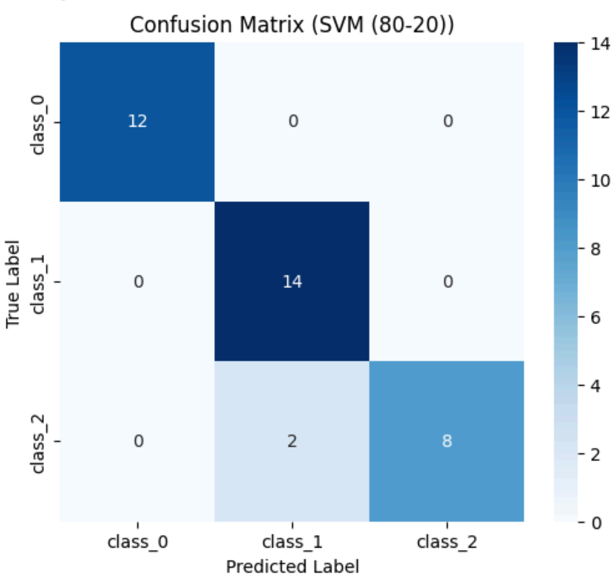

```
==================================================
 Train-Test Split 60-40
==================================================
Using best params for SVM: {'C': np.float64(1.0), 'degree': np.int64(3), 'gamma': 'auto', 'kernel': 'rbf'}

SVM (60-40) Results:
Accuracy=0.9861, Precision=0.9866, Recall=0.9861, F1=0.9860
```



```
==================================================
 Train-Test Split 70-30
==================================================
Using best params for SVM: {'C': np.float64(1.0), 'degree': np.int64(2), 'gamma': 'scale', 'kernel': 'rbf'}

SVM (70-30) Results:
Accuracy=0.9815, Precision=0.9823, Recall=0.9815, F1=0.9814
```




```
==================================================
 Train-Test Split 80-20
==================================================
Using best params for SVM: {'C': np.float64(10.0), 'degree': np.int64(3), 'gamma': 'auto', 'kernel': 'rbf'}

SVM (80-20) Results:
Accuracy=0.9444, Precision=0.9514, Recall=0.9444, F1=0.9432
```



6. Generate a summary DataFrame of performance metrics and best parameters, then visualize it as a styled table using Matplotlib

```
# Prepare DataFrame
summary_df = pd.DataFrame(results_summary,
```

```
                          columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

# Create table
table = ax.table(cellText=summary_df.round(3).astype(str).values,
                 colLabels=summary_df.columns,
                 cellLoc='center',
                 loc='center')

# Disable automatic font sizing and set font size
table.auto_set_font_size(False)
table.set_fontsize(10)

# Scale table: horizontal scale is arbitrary, we will set col widths explicitly
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
# sum should be ~1.0 for proportion

# Adjust each column width by modifying cell widths in each row (including header)
for col_idx, width in enumerate(col_widths):
    # Header cell
    cell = table[0, col_idx]
    cell.set_width(width)
    # Data cells
    for row_idx in range(1, len(summary_df) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("SVM Summary Across Splits (with Best Hyperparameters)", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

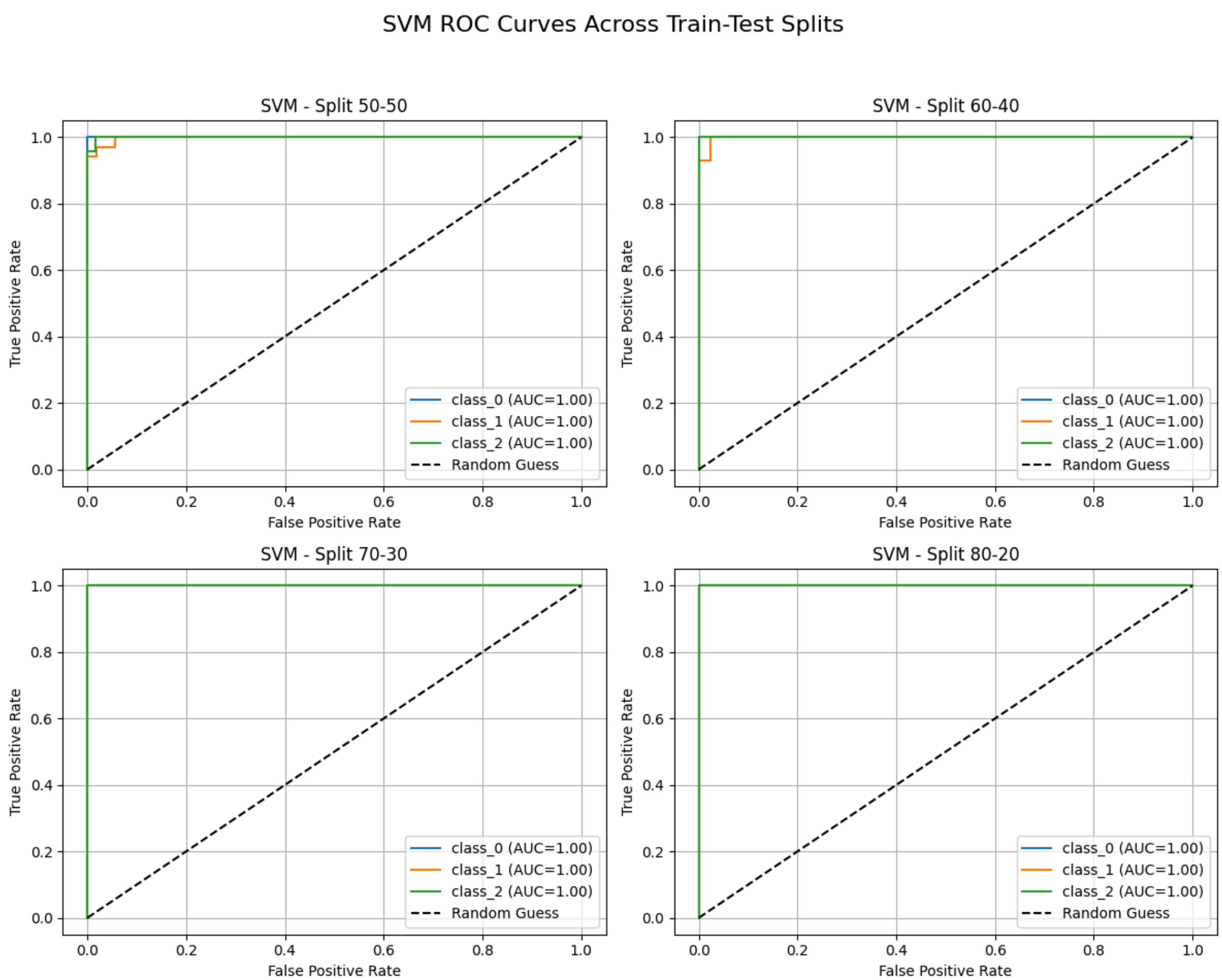
SVM Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.978	0.978	0.978	0.977	C=0.1, degree=2, gamma=scale, kernel=linear
60-40	0.986	0.987	0.986	0.986	C=1.0, degree=3, gamma=auto, kernel=rbf
70-30	0.981	0.982	0.981	0.981	C=1.0, degree=2, gamma=scale, kernel=rbf
80-20	0.944	0.951	0.944	0.943	C=10.0, degree=3, gamma=auto, kernel=rbf

7. Visualize multiclass ROC curves with AUC values across different train-test splits using subplots

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()
```

```python
for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"SVM - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("SVM ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



SVM with PCA

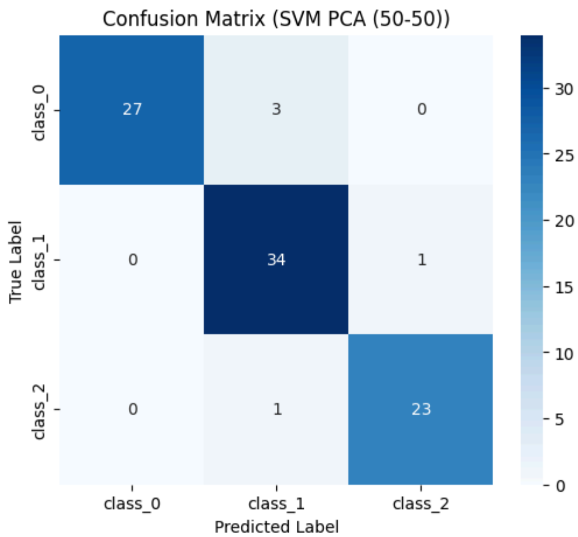
8. ApplyPCA for dimensionality reduction, then perform hyperparameter tuning and evaluation of SVM models across different train-test splits

```python
from sklearn.decomposition import PCA
```

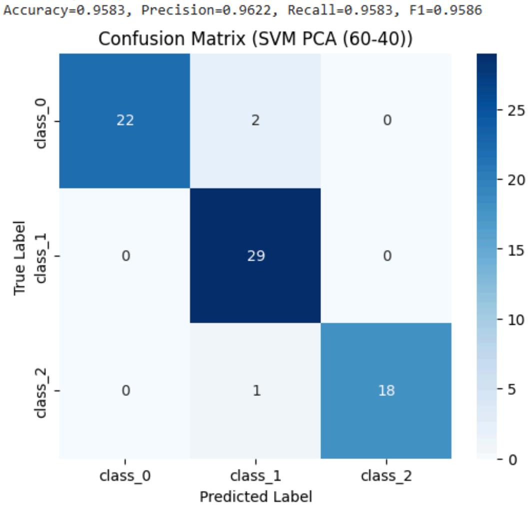
```python
# PCA with 5 components
pca = PCA(n_components=5)

splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_pca = []
roc_collector_pca = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label} with PCA\n{'='*50}")

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

    # PCA fit on train only!
    pca.fit(X_train)
    X_train_pca = pca.transform(X_train)
    X_test_pca = pca.transform(X_test)

    # Standardize after PCA (optional but often useful)
    scaler = StandardScaler()
    X_train_pca = scaler.fit_transform(X_train_pca)
    X_test_pca = scaler.transform(X_test_pca)

    # Halving Grid Search on PCA-transformed data
    grid = HalvingGridSearchCV(
        svm_model, svm_params, cv=5, scoring="accuracy",
        n_jobs=-1, random_state=42, verbose=0
    )
    grid.fit(X_train_pca, y_train)

    results = pd.DataFrame(grid.cv_results_)
    results = results.sort_values(by="mean_test_score", ascending=False)

    top_acc = results.iloc[0]["mean_test_score"]
    if top_acc >= 1.0:
        valid_results = results[results["mean_test_score"] < 1.0]
    else:
        valid_results = results

    if not valid_results.empty:
        best_row = valid_results.iloc[0]
        best_params = {k.replace("param_", ""): best_row[k]
                       for k in results.columns if k.startswith("param_")}
        print(f"Using best params for SVM with PCA: {best_params}")

        best_model = SVC(**best_params, probability=True, random_state=42)

        acc, prec, rec, f1, roc_data = evaluate_model(
```

```python
            best_model, X_train_pca, X_test_pca, y_train, y_test,
            model_name=f"SVM PCA ({split_label})"
        )
        params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
        results_summary_pca.append([split_label, acc, prec, rec, f1, params_str])

        if roc_data:
            roc_collector_pca[split_label] = roc_data

    else:
        print(f"No valid SVM models under 1.0 accuracy for split {split_label} with PCA")
```

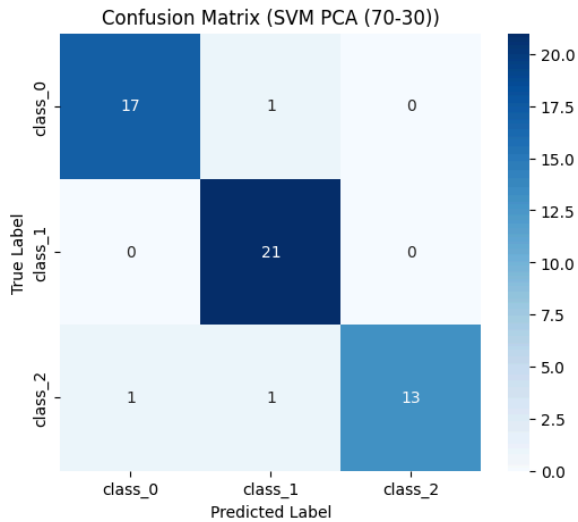
```
==================================================
 Train-Test Split 50-50 with PCA
==================================================
Using best params for SVM with PCA: {'C': np.float64(0.1), 'degree': np.int64(2), 'gamma': 'scale', 'kernel': 'linear'}

SVM PCA (50-50) Results:
Accuracy=0.9438, Precision=0.9474, Recall=0.9438, F1=0.9441
```



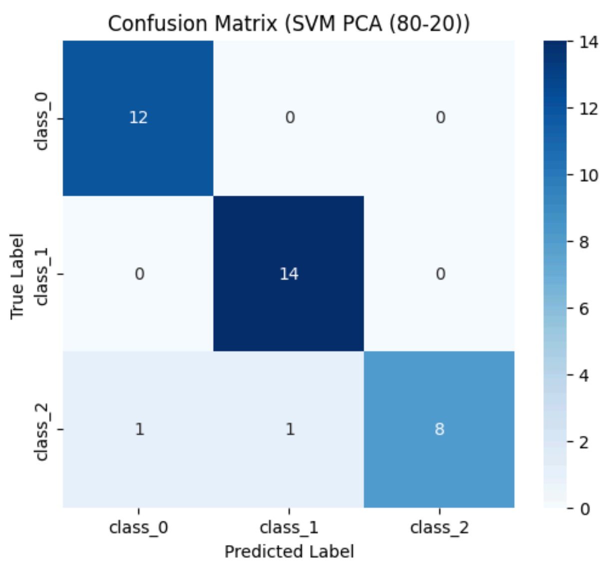

```
==================================================
 Train-Test Split 60-40 with PCA
==================================================
Using best params for SVM with PCA: {'C': np.float64(0.1), 'degree': np.int64(2), 'gamma': 'scale', 'kernel': 'linear'}

SVM PCA (60-40) Results:
Accuracy=0.9583, Precision=0.9622, Recall=0.9583, F1=0.9586
```



```
==================================================
 Train-Test Split 70-30 with PCA
==================================================
Using best params for SVM with PCA: {'C': np.float64(1.0), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'linear'}

SVM PCA (70-30) Results:
Accuracy=0.9444, Precision=0.9477, Recall=0.9444, F1=0.9440
```




```
==================================================
 Train-Test Split 80-20 with PCA
==================================================
Using best params for SVM with PCA: {'C': np.float64(1.0), 'degree': np.int64(3), 'gamma': 'auto', 'kernel': 'linear'}

SVM PCA (80-20) Results:
Accuracy=0.9444, Precision=0.9484, Recall=0.9444, F1=0.9424
```



9. Visualize performance metrics and best hyperparameters of SVM models trained on PCA-transformed data across multiple splits

```
# Summary Table for PCA
summary_df_pca = pd.DataFrame(results_summary_pca,
                              columns=["Split", "Accuracy", "Precision", "Recall", "F1",
"Best Hyperparameters"])
```

```
fig, ax = plt.subplots(figsize=(14, len(summary_df_pca) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_pca.round(3).astype(str).values,
                 colLabels=summary_df_pca.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]  # same as before

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_pca) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("SVM PCA Summary Across Splits (with Best Hyperparameters)", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

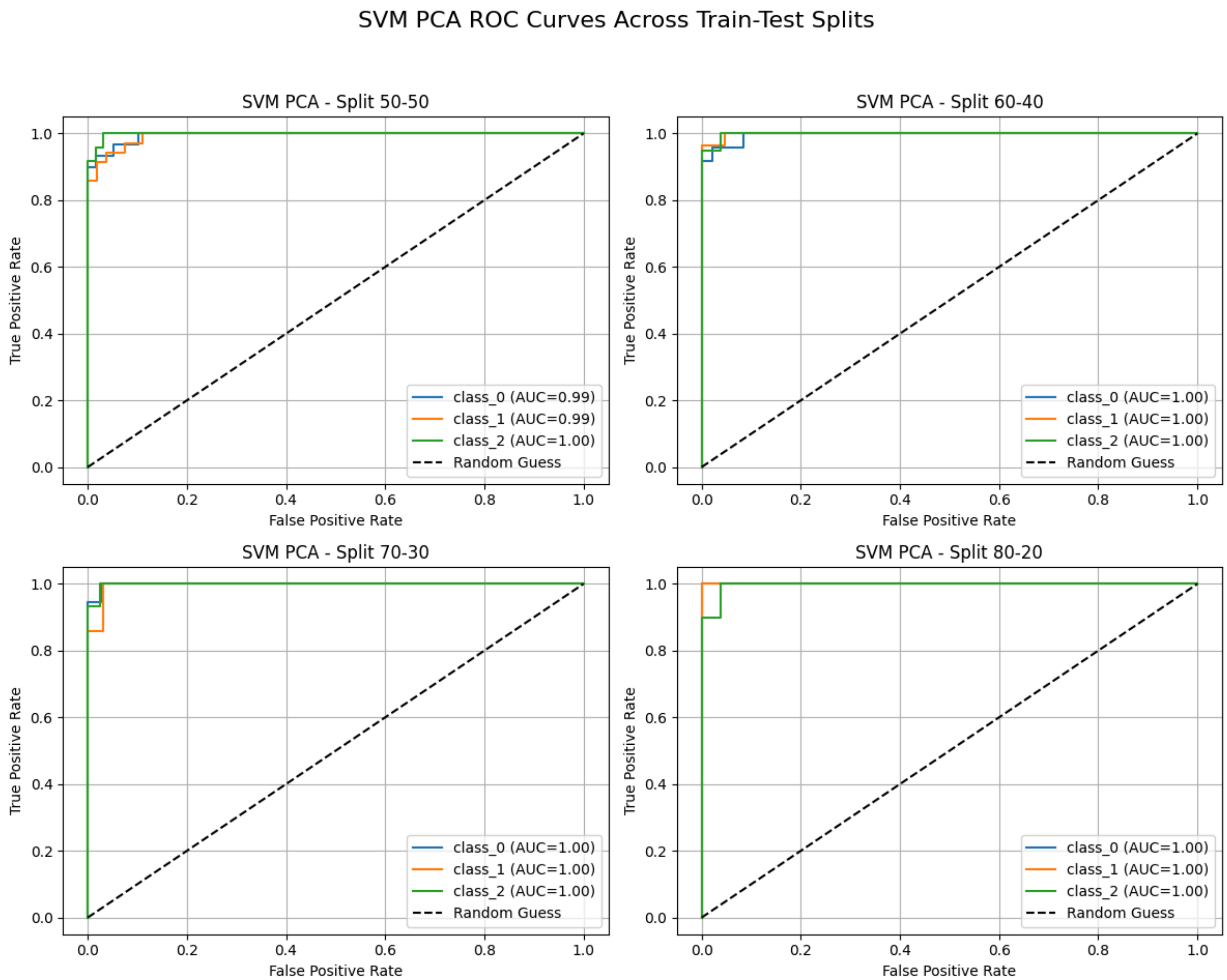
SVM PCA Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.944	0.947	0.944	0.944	C=0.1, degree=2, gamma=scale, kernel=linear
60-40	0.958	0.962	0.958	0.959	C=0.1, degree=2, gamma=scale, kernel=linear
70-30	0.944	0.948	0.944	0.944	C=1.0, degree=3, gamma=scale, kernel=linear
80-20	0.944	0.948	0.944	0.942	C=1.0, degree=3, gamma=auto, kernel=linear

10. Visualize ROC curves and AUC scores for each class across multiple train-test splits of SVM models with PCA preprocessing

```
# ROC curves for PCA runs
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_pca.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"SVM PCA - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
```

```
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("SVM PCA ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



MLP without PCA

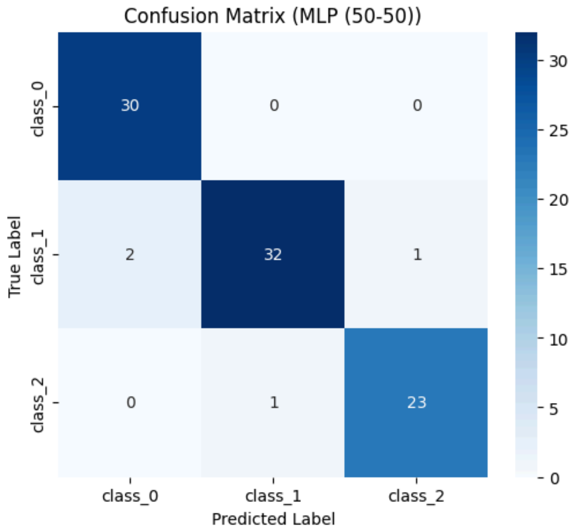
11. Define MLP Model and Hyperparameter Grid for Tuning

```
# MLP model and hyperparameter grid
mlp_model = MLPClassifier(random_state=42, max_iter=50)
mlp_params = {
    'hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001, 0.01],  # L2 regularization
    'learning_rate': ['constant', 'adaptive']
}
```

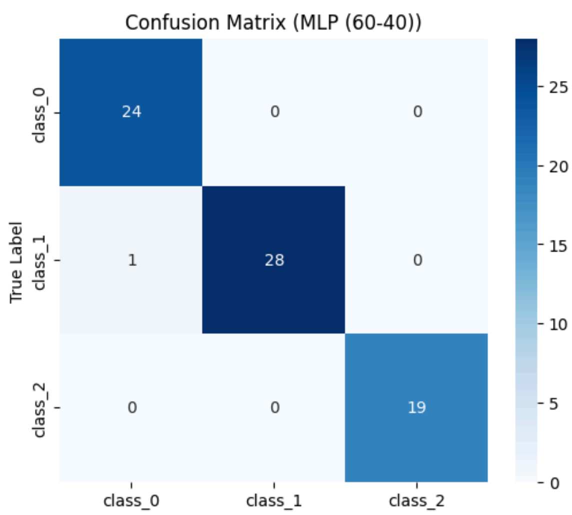
12. Initialize Multi-Layer Perceptron (MLP) model and specify hyperparameter search space

```
results_summary_mlp = []
roc_collector_mlp = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label} (MLP)\n{'='*50}")

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

    # Standardize features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Halving Grid Search for MLP
    grid = HalvingGridSearchCV(
        mlp_model, mlp_params, cv=5, scoring="accuracy",
        n_jobs=-1, random_state=42, verbose=0
    )
    grid.fit(X_train, y_train)

    results = pd.DataFrame(grid.cv_results_)
    results = results.sort_values(by="mean_test_score", ascending=False)

    top_acc = results.iloc[0]["mean_test_score"]
    if top_acc >= 1.0:
        valid_results = results[results["mean_test_score"] < 1.0]
    else:
        valid_results = results

    if not valid_results.empty:
        best_row = valid_results.iloc[0]
        best_params = {k.replace("param_", ""): best_row[k]
                       for k in results.columns if k.startswith("param_")}
        print(f"Using best params for MLP: {best_params}")

        best_model = MLPClassifier(**best_params, random_state=42, max_iter=1000)

        acc, prec, rec, f1, roc_data = evaluate_model(
            best_model, X_train, X_test, y_train, y_test,
            model_name=f"MLP ({split_label})"
        )
        params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
        results_summary_mlp.append([split_label, acc, prec, rec, f1, params_str])

        if roc_data:
            roc_collector_mlp[split_label] = roc_data

    else:
        print(f"No valid MLP models under 1.0 accuracy for split {split_label}")
```

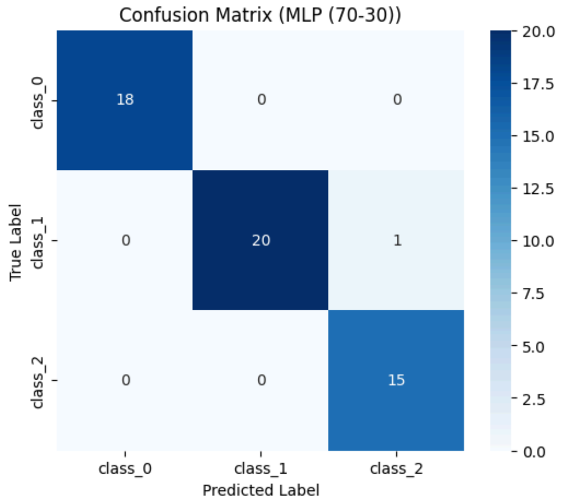
```
==================================================
 Train-Test Split 50-50 (MLP)
==================================================
Using best params for MLP: {'activation': 'relu', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive'}

MLP (50-50) Results:
Accuracy=0.9551, Precision=0.9558, Recall=0.9551, F1=0.9548
```



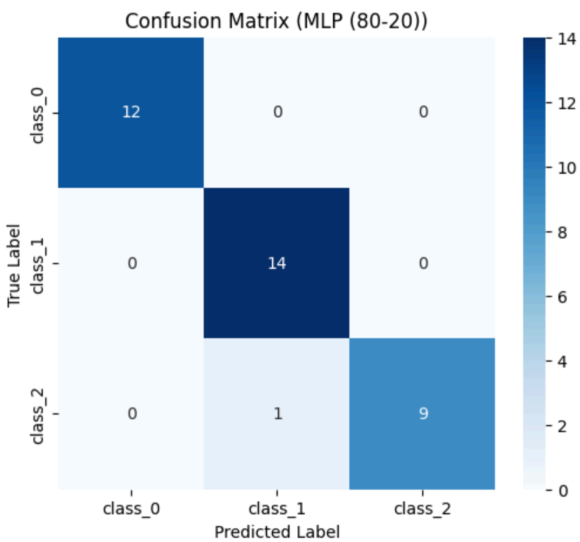

```
==================================================
 Train-Test Split 60-40 (MLP)
==================================================
Using best params for MLP: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP (60-40) Results:
Accuracy=0.9861, Precision=0.9867, Recall=0.9861, F1=0.9861
```



```
==================================================
 Train-Test Split 70-30 (MLP)
==================================================
Using best params for MLP: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (50, 50), 'learning_rate': 'adaptive'}

MLP (70-30) Results:
Accuracy=0.9815, Precision=0.9826, Recall=0.9815, F1=0.9816
```




```
======================================================
 Train-Test Split 80-20 (MLP)
======================================================
Using best params for MLP: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP (80-20) Results:
Accuracy=0.9722, Precision=0.9741, Recall=0.9722, F1=0.9720
```



13. Display MLP performance summary with best hyperparameters across splits

```
summary_df_mlp = pd.DataFrame(results_summary_mlp,
                              columns=["Split", "Accuracy", "Precision", "Recall", "F1",
"Best Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_mlp) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')
```

```
table = ax.table(cellText=summary_df_mlp.round(3).astype(str).values,
                 colLabels=summary_df_mlp.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_mlp) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("MLP Summary Across Splits (with Best Hyperparameters)", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

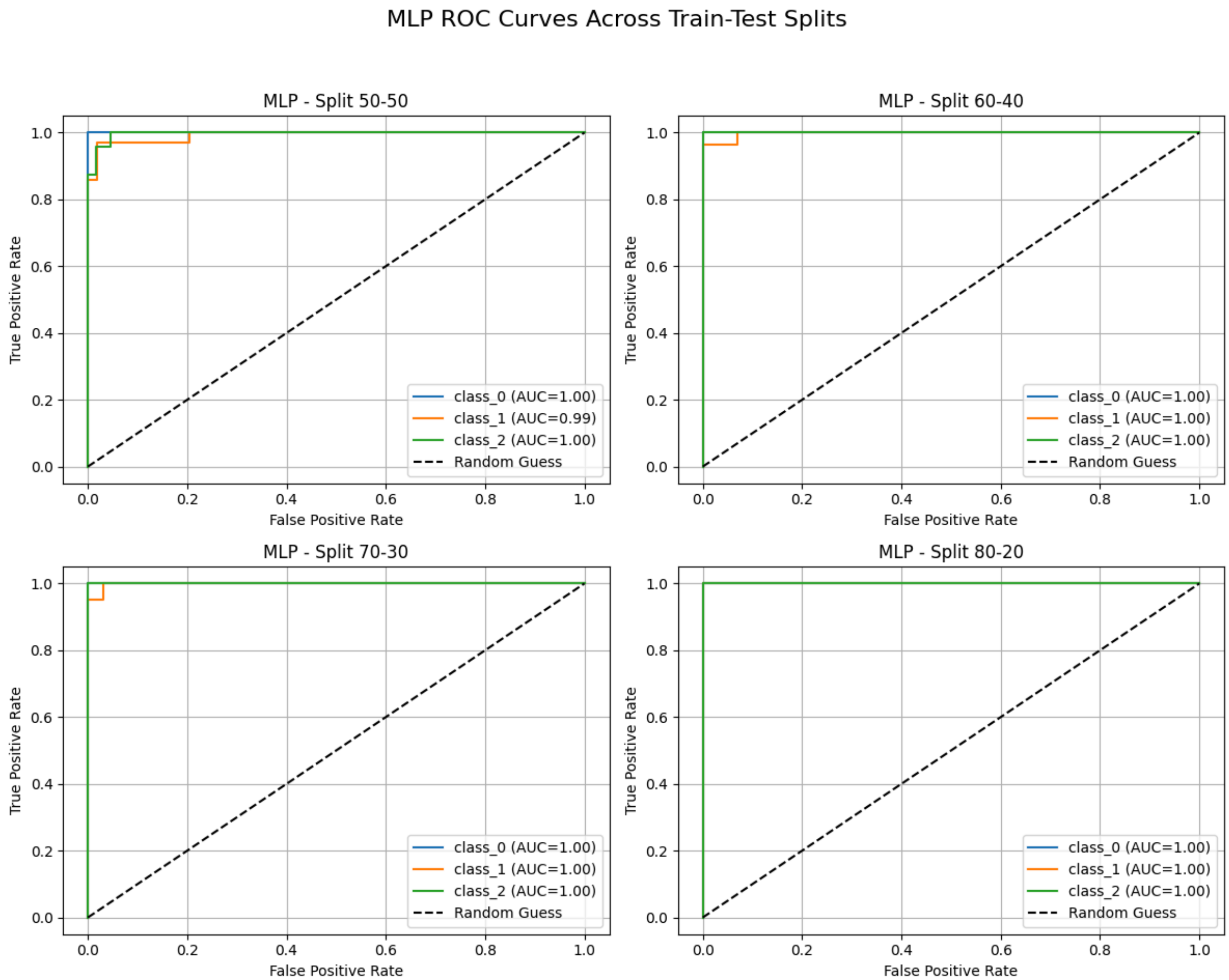
MLP Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.955	0.956	0.955	0.955	activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), learning_rate=adaptive
60-40	0.986	0.987	0.986	0.986	activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), learning_rate=constant
70-30	0.981	0.983	0.981	0.982	activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), learning_rate=adaptive
80-20	0.972	0.974	0.972	0.972	activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), learning_rate=constant

14. Plot ROC curves for MLP models across train-test splits

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_mlp.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"MLP - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("MLP ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
```

```
plt.show()
```



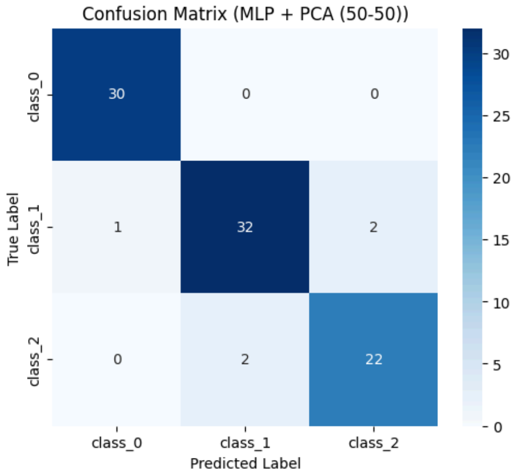
MLP with PCA

15. Train and evaluate MLP models with PCA dimensionality reduction across multiple splits

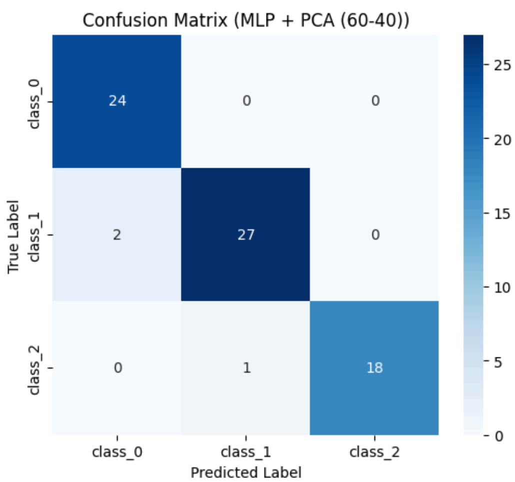
```
results_summary_mlp_pca = []
roc_collector_mlp_pca = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label} (MLP + PCA)\n{'='*50}")

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, train_size=train_size, test_size=test_size,
        random_state=42, stratify=y
    )

    # Scale features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)

# PCA transform
pca = PCA(n_components=5, random_state=42)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# HalvingGridSearch for MLP with PCA data
grid = HalvingGridSearchCV(
    mlp_model, mlp_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for MLP + PCA: {best_params}")

    best_model = MLPClassifier(**best_params, random_state=42, max_iter=1000)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train, X_test, y_train, y_test,
        model_name=f"MLP + PCA ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_mlp_pca.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_mlp_pca[split_label] = roc_data

else:
    print(f"No valid MLP models under 1.0 accuracy for split {split_label}")
```

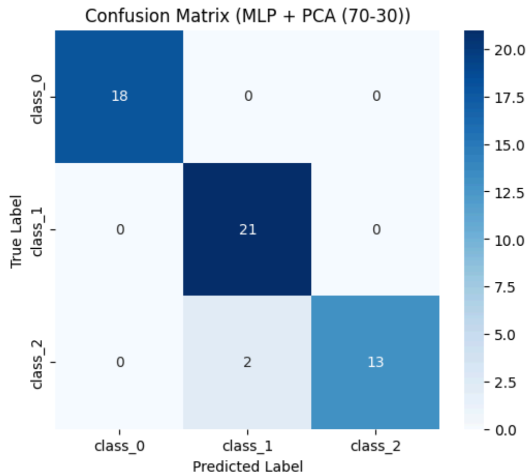
```
==================================================
 Train-Test Split 50-50 (MLP + PCA)
==================================================
Using best params for MLP + PCA: {'activation': 'relu', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (50,), 'learning_rate': 'constant'}

MLP + PCA (50-50) Results:
Accuracy=0.9438, Precision=0.9435, Recall=0.9438, F1=0.9435
```



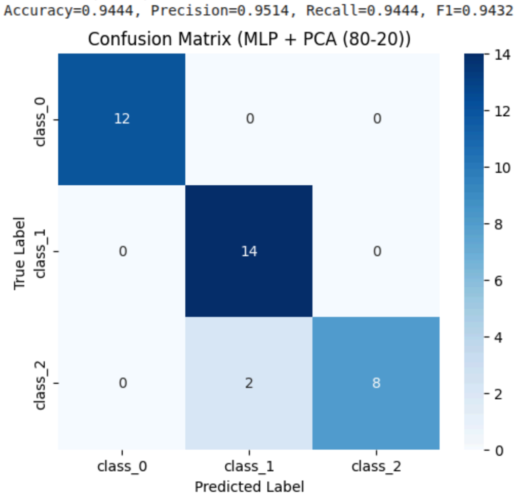

```
==================================================
 Train-Test Split 60-40 (MLP + PCA)
==================================================
Using best params for MLP + PCA: {'activation': 'tanh', 'alpha': np.float64(0.001), 'hidden_layer_sizes': (50,), 'learning_rate': 'adaptive'}

MLP + PCA (60-40) Results:
Accuracy=0.9583, Precision=0.9600, Recall=0.9583, F1=0.9583
```



```
==================================================
 Train-Test Split 70-30 (MLP + PCA)
==================================================
Using best params for MLP + PCA: {'activation': 'relu', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (50,), 'learning_rate': 'constant'}

MLP + PCA (70-30) Results:
Accuracy=0.9630, Precision=0.9662, Recall=0.9630, F1=0.9625
```




```
==================================================
 Train-Test Split 80-20 (MLP + PCA)
==================================================
Using best params for MLP + PCA: {'activation': 'relu', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (50,), 'learning_rate': 'constant'}

MLP + PCA (80-20) Results:
Accuracy=0.9444, Precision=0.9514, Recall=0.9444, F1=0.9432
```



16. Generate summary table for MLP models with PCA across different train-test splits

```
# Prepare DataFrame and plot summary table
summary_df_mlp_pca = pd.DataFrame(results_summary_mlp_pca,
                                  columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_mlp_pca) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_mlp_pca.round(3).astype(str).values,
                 colLabels=summary_df_mlp_pca.columns,
```

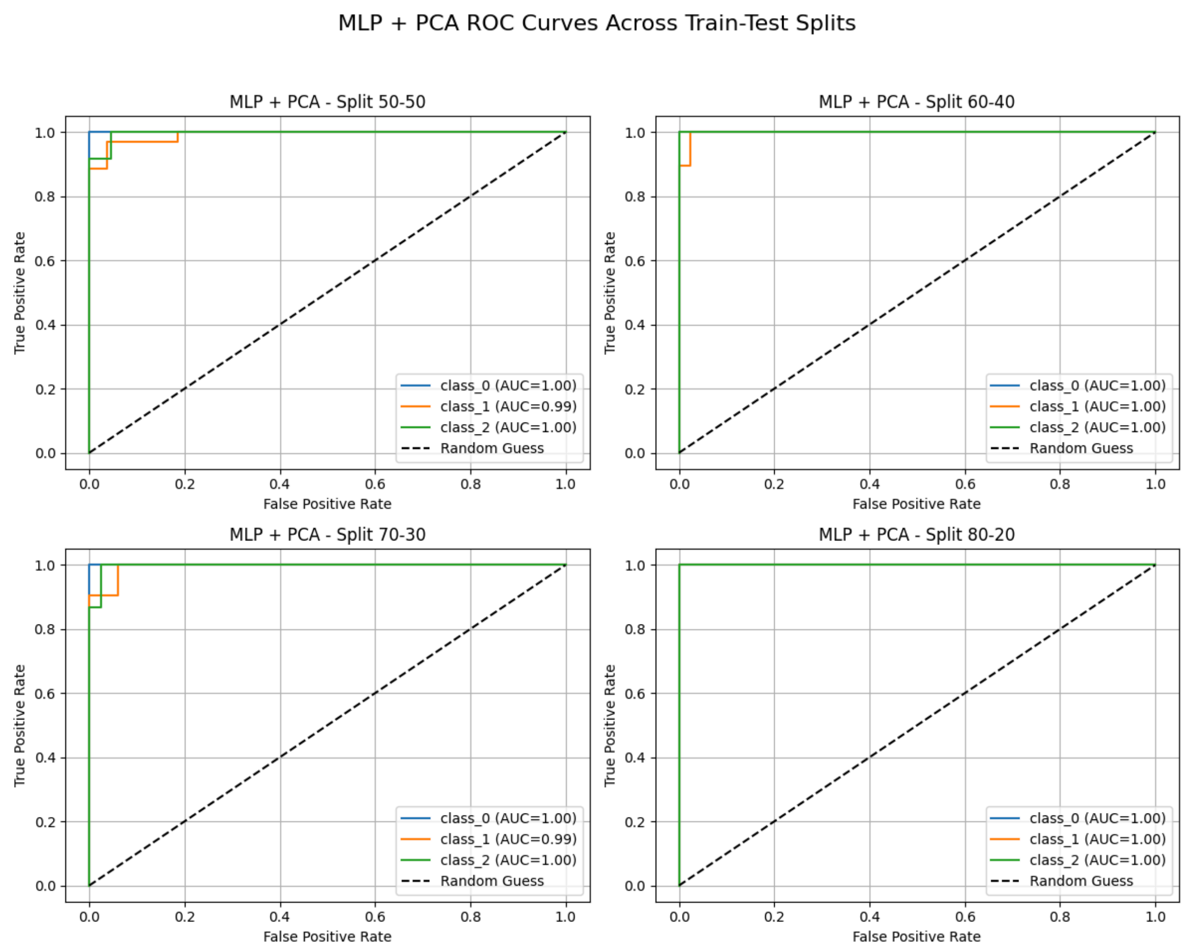
```
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_mlp_pca) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("MLP + PCA Summary Across Splits (with Best Hyperparameters)", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

MLP + PCA Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.944	0.944	0.944	0.944	activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), learning_rate=constant
60-40	0.958	0.96	0.958	0.958	activation=tanh, alpha=0.001, hidden_layer_sizes=(50,), learning_rate=adaptive
70-30	0.963	0.966	0.963	0.962	activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), learning_rate=constant
80-20	0.944	0.951	0.944	0.943	activation=relu, alpha=0.0001, hidden_layer_sizes=(50,), learning_rate=constant

17. Plot ROC curves for MLP + PCA models across train-test splits

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_mlp_pca.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"MLP + PCA - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("MLP + PCA ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Random Forest without PCA

18. Define Random Forest Model and Hyperparameter Grid

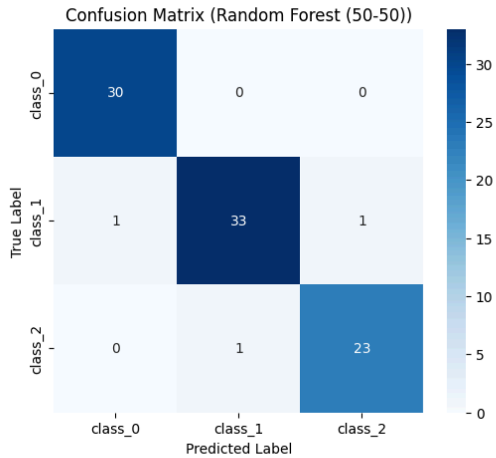
```
from sklearn.ensemble import RandomForestClassifier

# Random Forest model and params
rf_model = RandomForestClassifier(random_state=42)
rf_params = {
    'n_estimators': [10, 15, 30],
    'max_depth': [None, 2, 5],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [1, 2]
}
```

19. Train and Evaluate Random Forest Models Across Multiple Train-Test Splits

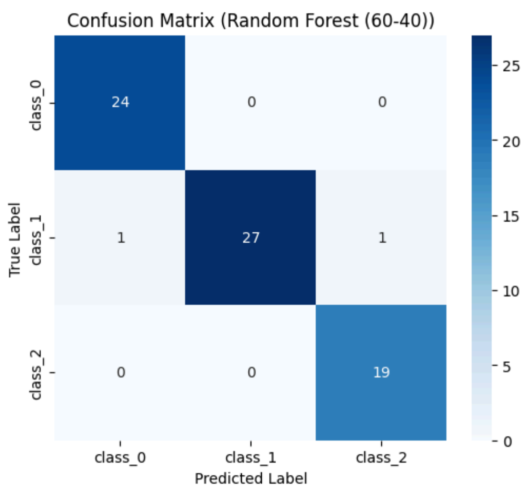
```
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_rf = []
roc_collector_rf = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label} (Random Forest)\n{'='*50}")
```

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

# Standardize features (optional for RF, but for consistency)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

grid = HalvingGridSearchCV(
    rf_model, rf_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for RF: {best_params}")

    best_model = RandomForestClassifier(**best_params, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train, X_test, y_train, y_test,
        model_name=f"Random Forest ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_rf.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_rf[split_label] = roc_data

else:
    print(f"No valid RF models under 1.0 accuracy for split {split_label}")
```

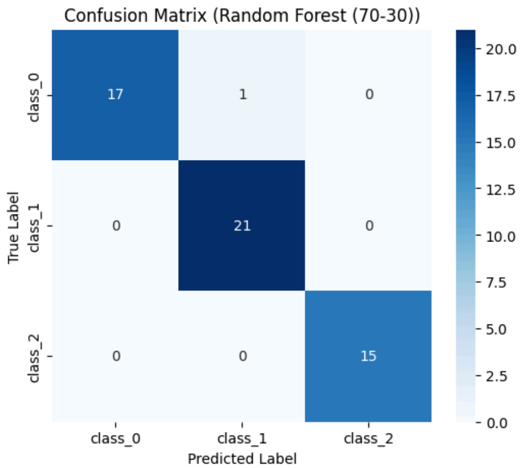
```
==================================================
 Train-Test Split 50-50 (Random Forest)
==================================================
Using best params for RF: {'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(10)}

Random Forest (50-50) Results:
Accuracy=0.9663, Precision=0.9663, Recall=0.9663, F1=0.9661
```



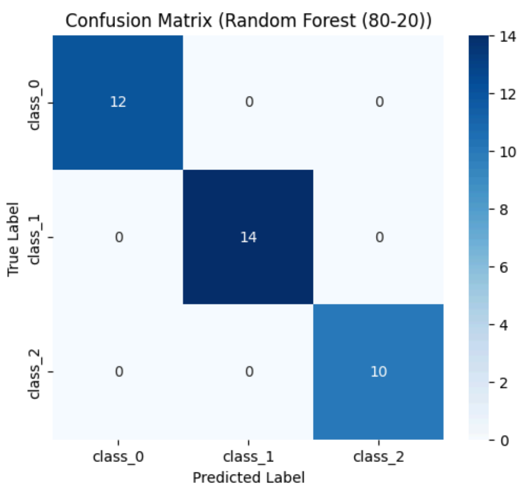

```
==================================================
 Train-Test Split 60-40 (Random Forest)
==================================================
Using best params for RF: {'max_depth': 5, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(30)}

Random Forest (60-40) Results:
Accuracy=0.9722, Precision=0.9735, Recall=0.9722, F1=0.9720
```



```
==================================================
 Train-Test Split 70-30 (Random Forest)
==================================================
Using best params for RF: {'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(15)}

Random Forest (70-30) Results:
Accuracy=0.9815, Precision=0.9823, Recall=0.9815, F1=0.9814
```




```
==================================================
 Train-Test Split 80-20 (Random Forest)
==================================================
Using best params for RF: {'max_depth': 5, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(3), 'n_estimators': np.int64(30)}

Random Forest (80-20) Results:
Accuracy=1.0000, Precision=1.0000, Recall=1.0000, F1=1.0000
```



20. Generate and Display Summary Table for Random Forest Models Across Train-Test Splits

```
# Prepare summary table
summary_df_rf = pd.DataFrame(results_summary_rf,
                             columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best 
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_rf) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_rf.round(3).astype(str).values,
                 colLabels=summary_df_rf.columns,
                 cellLoc='center',
```

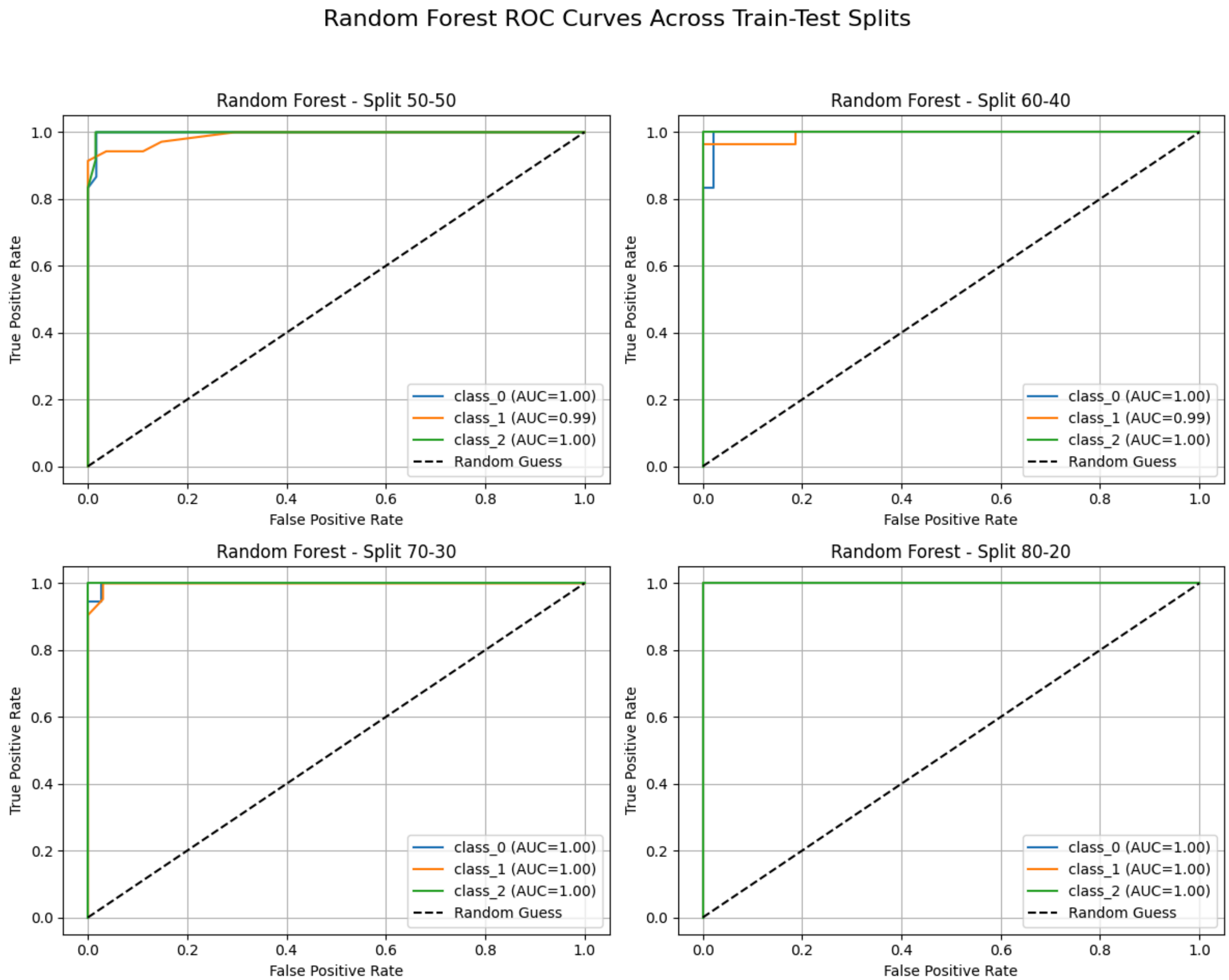
```
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_rf) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("Random Forest Summary Across Splits (with Best Hyperparameters)", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

Random Forest Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.966	0.966	0.966	0.966	max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=10
60-40	0.972	0.973	0.972	0.972	max_depth=5, min_samples_leaf=1, min_samples_split=2, n_estimators=30
70-30	0.981	0.982	0.981	0.981	max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=15
80-20	1.0	1.0	1.0	1.0	max_depth=5, min_samples_leaf=1, min_samples_split=3, n_estimators=30

21. Plot ROC Curves for Random Forest Models Across Train-Test Splits

```
# ROC curves plot
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_rf.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"Random Forest - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("Random Forest ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Random Forest with PCA

22. Define Random Forest Hyperparameters for Models with PCA

```
rf_params = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'criterion': ['gini']
}
```

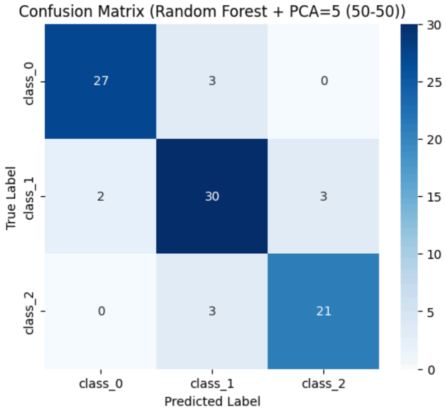
23. Train and Evaluate Random Forest Models with PCA (5 Components) Across Train-Test Splits

```
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_rf = []
roc_collector_rf = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n Train-Test Split {split_label} (Random Forest + PCA=5)\n{'='*50}")
```

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

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA with 5 components
pca = PCA(n_components=5, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Hyperparameter tuning with HalvingGridSearchCV on PCA-transformed data
grid = HalvingGridSearchCV(
    rf_model, rf_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train_pca, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for RF: {best_params}")

    best_model = RandomForestClassifier(**best_params, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train_pca, X_test_pca, y_train, y_test,
        model_name=f"Random Forest + PCA=5 ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_rf.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_rf[split_label] = roc_data

else:
    print(f"No valid RF models under 1.0 accuracy for split {split_label}")
```

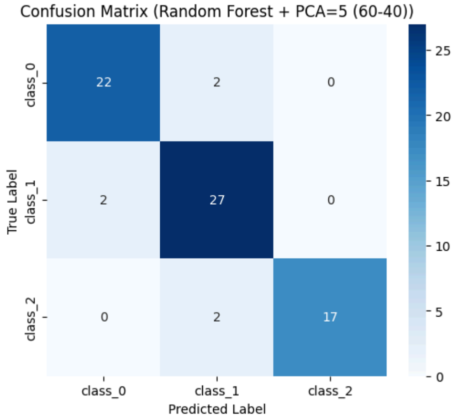
```
==================================================
 Train-Test Split 50-50 (Random Forest + PCA=5)
==================================================
Using best params for RF: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(50)}

Random Forest + PCA=5 (50-50) Results:
Accuracy=0.8764, Precision=0.8775, Recall=0.8764, F1=0.8768
```

Confusion Matrix (Random Forest + PCA=5 (50-50))



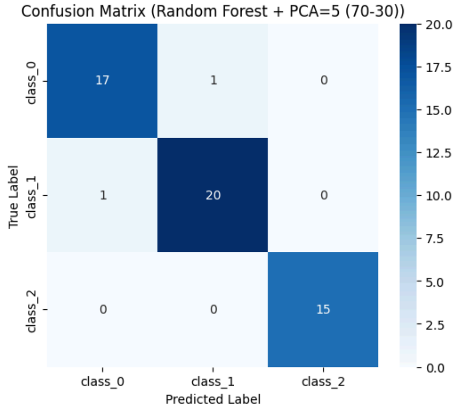

```
==================================================
 Train-Test Split 60-40 (Random Forest + PCA=5)
==================================================
Using best params for RF: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(50)}

Random Forest + PCA=5 (60-40) Results:
Accuracy=0.9167, Precision=0.9203, Recall=0.9167, F1=0.9173
```

Confusion Matrix (Random Forest + PCA=5 (60-40))

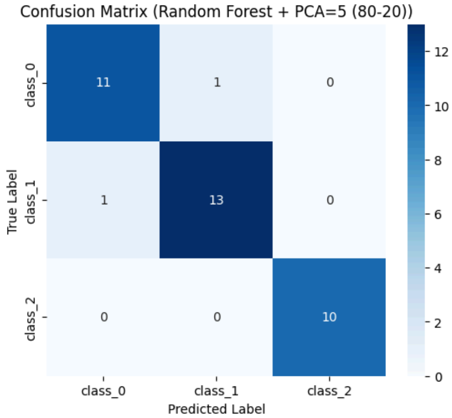



```
==================================================
 Train-Test Split 70-30 (Random Forest + PCA=5)
==================================================
Using best params for RF: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': np.int64(2), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(100)}

Random Forest + PCA=5 (70-30) Results:
Accuracy=0.9630, Precision=0.9630, Recall=0.9630, F1=0.9630
```

Confusion Matrix (Random Forest + PCA=5 (70-30))



```
==================================================
 Train-Test Split 80-20 (Random Forest + PCA=5)
==================================================
Using best params for RF: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(50)}

Random Forest + PCA=5 (80-20) Results:
Accuracy=0.9444, Precision=0.9444, Recall=0.9444, F1=0.9444
```



24. Generate and Display Summary Table for Random Forest + PCA (5 Components) Across Train-Test Splits

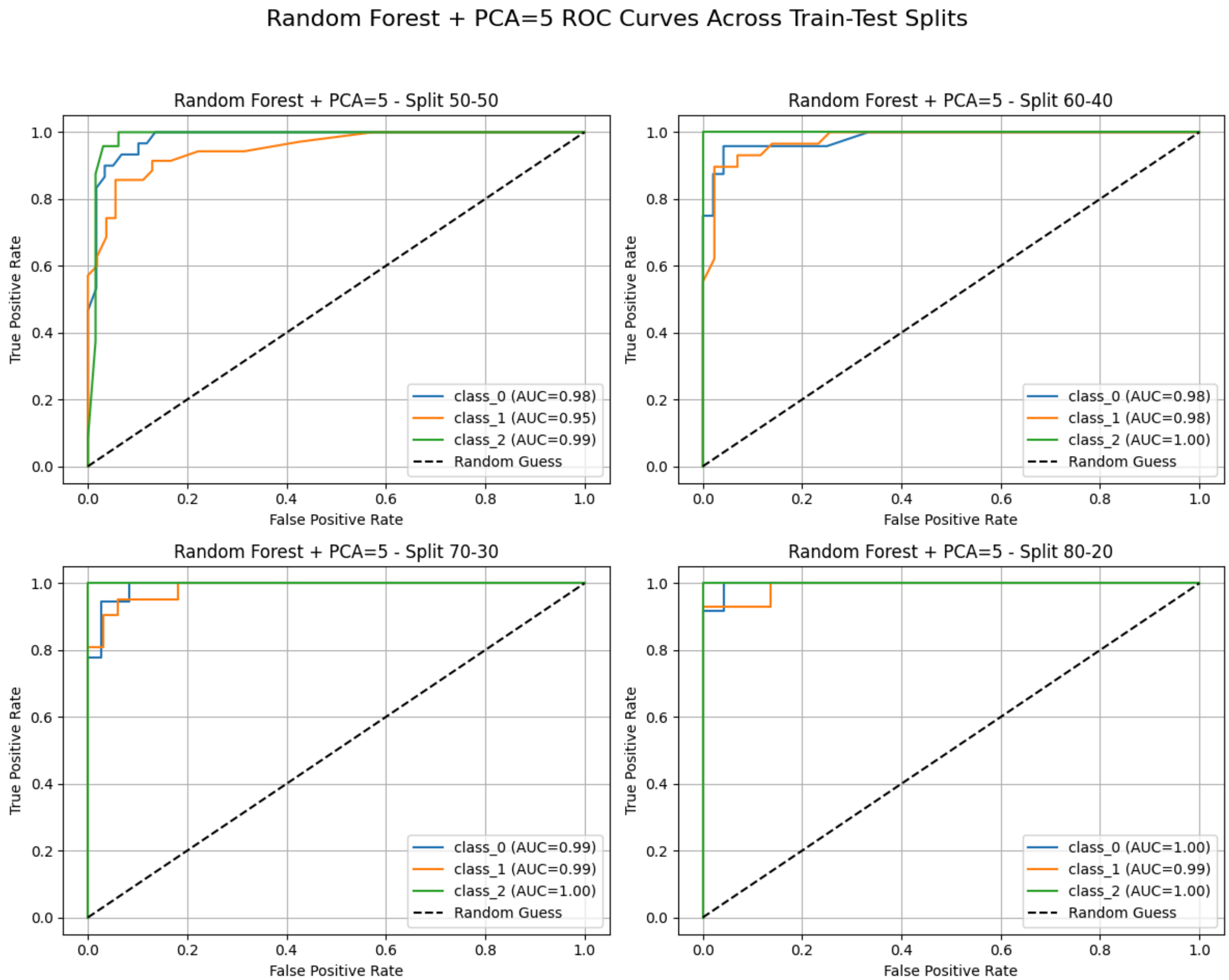
```
# Prepare summary table
summary_df_rf = pd.DataFrame(results_summary_rf,
                             columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_rf) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_rf.round(3).astype(str).values,
                 colLabels=summary_df_rf.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_rf) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("Random Forest + PCA=5 Summary Across Splits (with Best Hyperparameters)", fontsize=14,
pad=15)
plt.tight_layout()
plt.show()
```

Random Forest + PCA=5 Summary Across Splits (with Best Hyperparameters)

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.876	0.878	0.876	0.877	criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=50
60-40	0.917	0.92	0.917	0.917	criterion=gini, max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=50
70-30	0.963	0.963	0.963	0.963	criterion=gini, max_depth=None, min_samples_leaf=2, min_samples_split=5, n_estimators=100
80-20	0.944	0.944	0.944	0.944	criterion=gini, max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=50

25. Plot ROC Curves for Random Forest + PCA (5 Components) Models Across Train-Test Splits

```
# ROC curves plot
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_rf.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"Random Forest + PCA=5 - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("Random Forest + PCA=5 ROC Curves Across Train-Test Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



26. Discussion on Wine dataset:

The Wine dataset, a classic benchmark in machine learning, contains 178 samples with 13 continuous chemical features describing different cultivars of wine. The dataset is relatively balanced across three classes but presents overlapping distributions, making classification non-trivial. Its moderate dimensionality allows for testing both linear and non-linear classifiers, with or without dimensionality reduction.

Support Vector Machine (SVM)

The SVM classifier performed strongly across all splits, achieving accuracies ranging from 94.4% to 98.6%, with consistently high precision, recall, and F1-scores. Linear and RBF kernels were frequently selected as optimal, depending on the train-test split. SVM's strength lies in its ability to handle high-dimensional spaces and separate classes with maximum margins, which aligns well with the Wine dataset's structure. However, after applying PCA, performance slightly declined, with accuracy dropping by around 2–3% in most cases. This indicates that the original feature space preserved more discriminative information than the reduced one, and dimensionality reduction was not strictly beneficial.

Multilayer Perceptron (MLP)
The MLP classifier also achieved competitive results, with accuracies between 95.5% and 98.6%, comparable to SVM. Best hyperparameters often included small hidden layers (50–100 neurons) and either relu or tanh activations. The model benefitted from sufficient training data and captured non-linear relationships in the dataset effectively. However, with PCA, MLP performance decreased slightly (down to ~94–96% accuracy), mirroring the trend seen with SVM. While PCA improved computational efficiency by reducing dimensions, it likely discarded subtle feature interactions important for the MLP's representation learning.

Random Forest (RF)
Random Forest also proved highly effective, achieving accuracies between 96.6% and 100% across splits. Particularly in the 80-20 split, RF achieved perfect classification with precision, recall, and F1 all equal to 1.0. This robustness reflects RF's ensemble nature, combining multiple decision trees and reducing overfitting risks. Its ability to naturally handle feature interactions without preprocessing makes it especially well-suited for the Wine dataset. However, applying PCA significantly reduced performance, especially at lower train-test splits (e.g., ~87.6% at 50-50 split). Since Random Forest does not rely on linear separability, PCA's compression likely removed key discriminative signals.

Overall Comparison
Across models, Random Forest (without PCA) consistently outperformed others, even reaching perfect accuracy in some cases. SVM and MLP were close competitors, with slight differences in performance depending on splits, while PCA tended to reduce classification effectiveness across all models. This suggests that the Wine dataset's full feature set is already highly informative and

benefits from models that exploit complex, non-linear relationships rather than dimensionality reduction.

Conclusion

The evaluation highlights how model selection and preprocessing interact with dataset characteristics. For the Wine dataset, ensemble-based methods like Random Forest excel due to their ability to capture feature interactions and reduce overfitting. SVM and MLP also demonstrate strong performance, but their reliance on PCA reduced effectiveness slightly. Overall, Random Forest emerges as the most reliable and accurate model for this dataset, followed closely by SVM and MLP without PCA.

Digits Dataset

27. Dataset_Download

```
[ ] from sklearn.datasets import load_digits

    # Load Digits dataset
    digits = load_digits()
    X = digits.data
    y = digits.target
    classes = digits.target_names
    print(classes)
```

```
[0 1 2 3 4 5 6 7 8 9]
```

SVM without PCA

28. Function to train the model, compute accuracy, precision, recall, F1 score, plot confusion matrix and ROC curves if available

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import load_wine
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.svm import SVC
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import HalvingGridSearchCV
```

```python
from sklearn.metrics import (
    accuracy_score, precision_score, recall_score, f1_score,
    confusion_matrix, roc_curve, auc
)
def evaluate_model(model, X_train, X_test, y_train, y_test, model_name="Model"):
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)
    prec = precision_score(y_test, y_pred, average="weighted", zero_division=0)
    rec = recall_score(y_test, y_pred, average="weighted", zero_division=0)
    f1 = f1_score(y_test, y_pred, average="weighted", zero_division=0)
    print(f"\n{model_name} Results:")
    print(f"Accuracy={acc:.4f}, Precision={prec:.4f}, Recall={rec:.4f}, F1={f1:.4f}")
    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
                xticklabels=classes, yticklabels=classes)
    plt.title(f"Confusion Matrix ({model_name})")
    plt.xlabel("Predicted Label")
    plt.ylabel("True Label")
    plt.show()
    roc_data = None
    if hasattr(model, "predict_proba"):
        y_bin = label_binarize(y_test, classes=np.arange(len(classes)))
        y_score = model.predict_proba(X_test)

        fpr, tpr, roc_auc = {}, {}, {}
        for i in range(len(classes)):
            fpr[i], tpr[i], _ = roc_curve(y_bin[:, i], y_score[:, i])
            roc_auc[i] = auc(fpr[i], tpr[i])
        roc_data = (fpr, tpr, roc_auc)

    return acc, prec, rec, f1, roc_data
```

Define SVM classifier and hyperparameter search space for grid tuning

```python
svm_model = SVC(probability=True, random_state=42)
svm_params = {
    "kernel": ["linear", "poly", "rbf", "sigmoid"],
    "C": [0.1, 1, 10],
    "degree": [2, 3],
    "gamma": ["scale", "auto"]
}
```

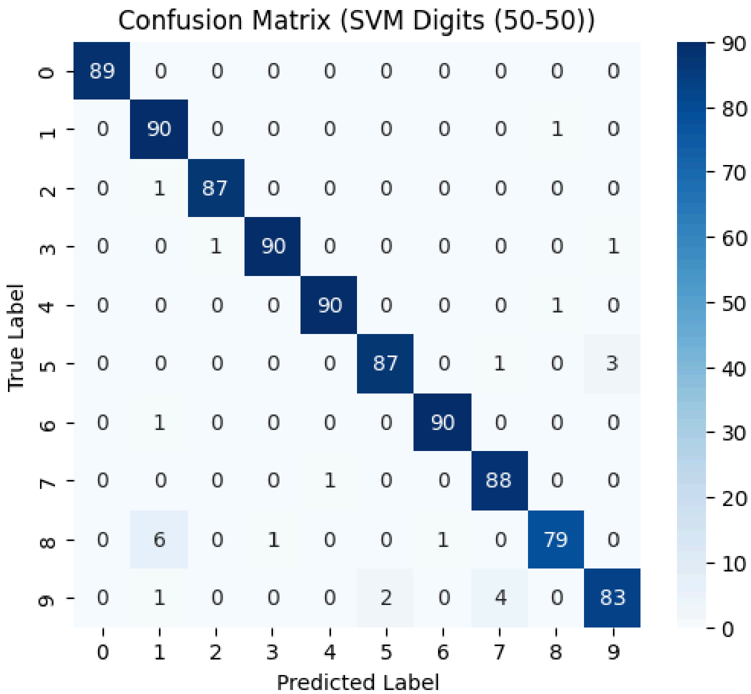
29. Perform train-test splits, standardize data, tune SVM hyperparameters with HalvingGridSearchCV, evaluate metrics, and collect ROC data

```python
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary = []
```

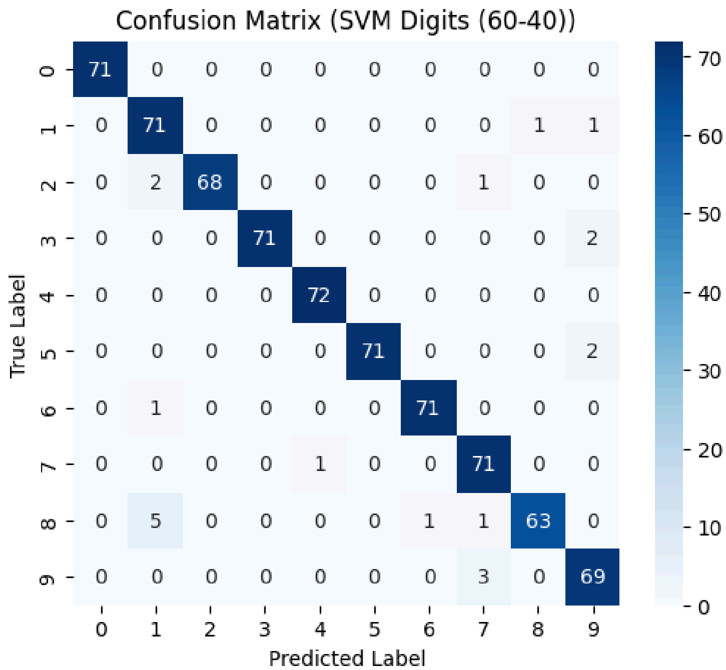
```python
roc_collector = {}

# Loop for SVM
for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n SVM on Digits — Split {split_label}\n{'='*50}")

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

    # Standardize features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Halving Grid Search
    grid = HalvingGridSearchCV(
        svm_model, svm_params, cv=5, scoring="accuracy",
        n_jobs=-1, random_state=42, verbose=0
    )
    grid.fit(X_train, y_train)

    results = pd.DataFrame(grid.cv_results_)
    results = results.sort_values(by="mean_test_score", ascending=False)

    top_acc = results.iloc[0]["mean_test_score"]
    if top_acc >= 1.0:
        valid_results = results[results["mean_test_score"] < 1.0]
    else:
        valid_results = results

    if not valid_results.empty:
        best_row = valid_results.iloc[0]
        best_params = {k.replace("param_", ""): best_row[k]
                       for k in results.columns if k.startswith("param_")}
        print(f"Using best params for SVM: {best_params}")

        best_model = SVC(**best_params, probability=True, random_state=42)

        acc, prec, rec, f1, roc_data = evaluate_model(
            best_model, X_train, X_test, y_train, y_test,
            model_name=f"SVM Digits ({split_label})"
        )
        params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
        results_summary.append([split_label, acc, prec, rec, f1, params_str])


        if roc_data:
            roc_collector[split_label] = roc_data
```

```
    else:
        print(f"No valid SVM models under 1.0 accuracy for split {split_label}")
```

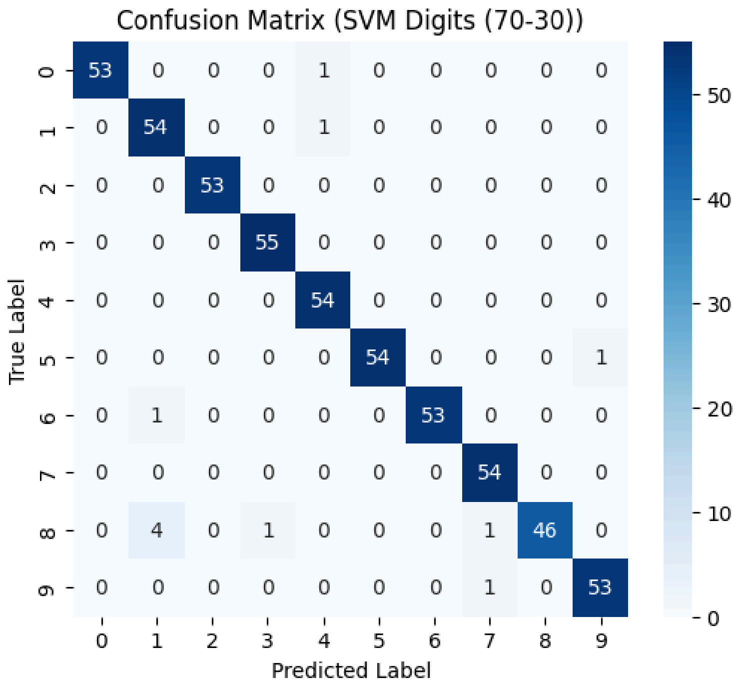
```
==================================================
 SVM on Digits – Split 50-50
==================================================
Best params: {'C': np.float64(0.1), 'degree': np.int64(3), 'gamma': 'auto', 'kernel': 'linear'}

SVM Digits (50-50) Results:
Accuracy=0.9711, Precision=0.9717, Recall=0.9711, F1=0.9710
```



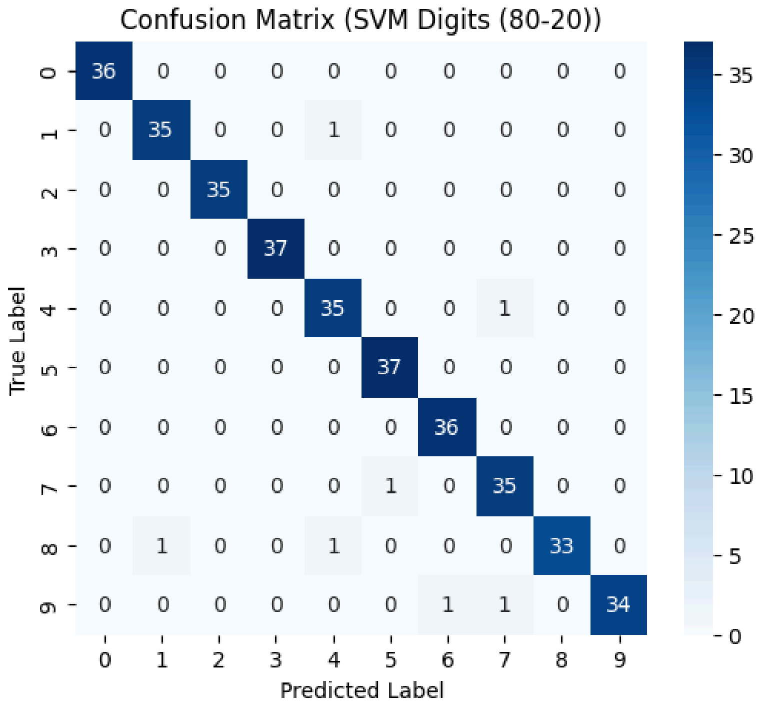
```
==================================================
 SVM on Digits — Split 60-40
==================================================
Best params: {'C': np.float64(10.0), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'linear'}

SVM Digits (60-40) Results:
Accuracy=0.9708, Precision=0.9721, Recall=0.9708, F1=0.9709
```



```
==================================================
 SVM on Digits — Split 70-30
==================================================
Best params: {'C': np.float64(0.1), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'linear'}

SVM Digits (70-30) Results:
Accuracy=0.9796, Precision=0.9806, Recall=0.9796, F1=0.9795
```



```
==================================================
 SVM on Digits – Split 80-20
==================================================
Best params: {'C': np.float64(10.0), 'degree': np.int64(2), 'gamma': 'auto', 'kernel': 'rbf'}

SVM Digits (80-20) Results:
Accuracy=0.9806, Precision=0.9810, Recall=0.9806, F1=0.9805
```



30. Generate a summary DataFrame of performance metrics and best parameters, then visualize it as a styled table using Matplotlib

```
# Prepare DataFrame
summary_df = pd.DataFrame(results_summary,
                          columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best 
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

# Create table
table = ax.table(cellText=summary_df.round(3).astype(str).values,
                 colLabels=summary_df.columns,
                 cellLoc='center',
                 loc='center')

# Disable automatic font sizing and set font size
table.auto_set_font_size(False)
table.set_fontsize(10)
```

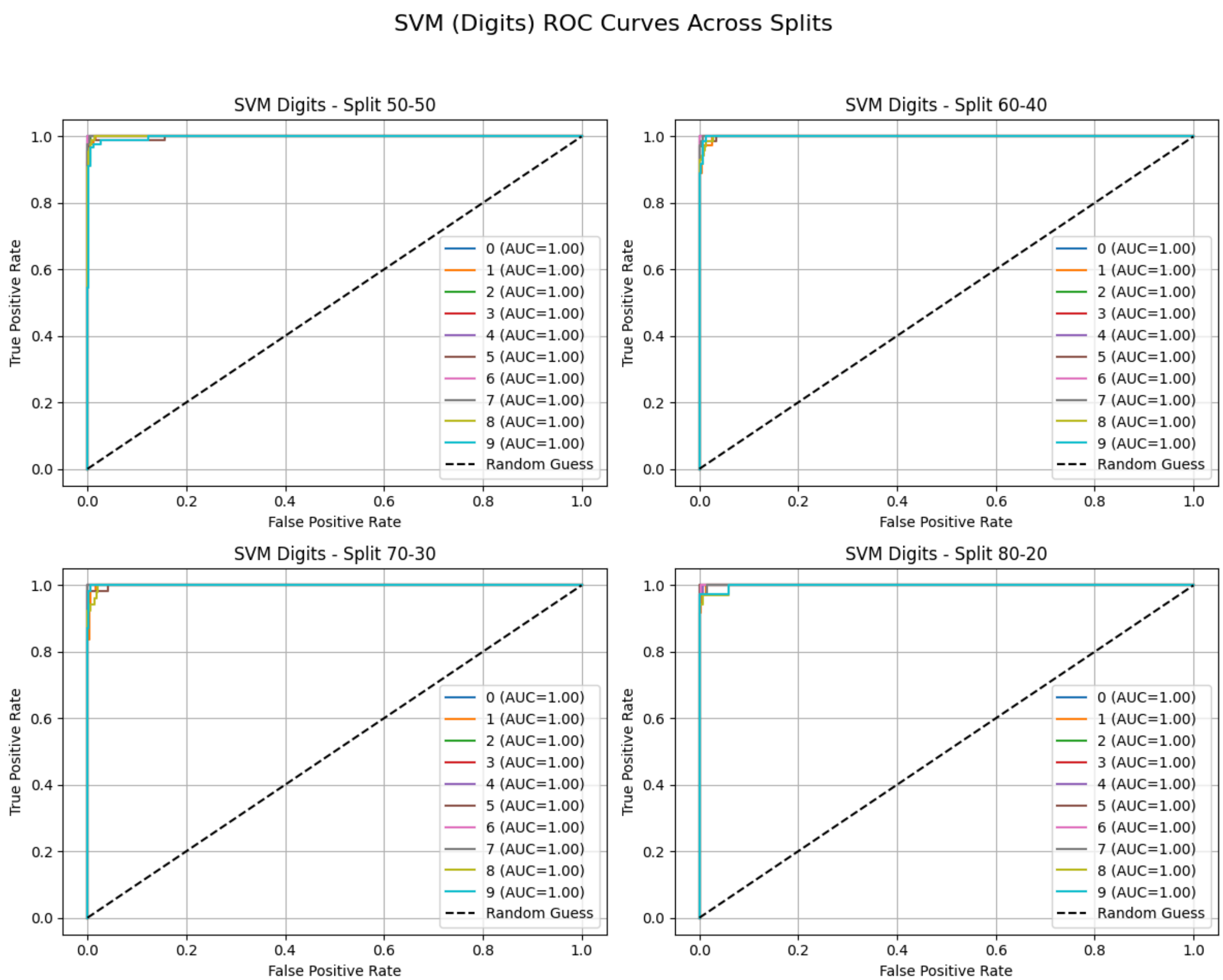
```
# Scale table: horizontal scale is arbitrary, we will set col widths explicitly
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
# sum should be ~1.0 for proportion

# Adjust each column width by modifying cell widths in each row (including header)
for col_idx, width in enumerate(col_widths):
    # Header cell
    cell = table[0, col_idx]
    cell.set_width(width)
    # Data cells
    for row_idx in range(1, len(summary_df) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("SVM (Digits) Summary Across Splits", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

SVM (Digits) Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.971	0.972	0.971	0.971	C=0.1, degree=3, gamma=auto, kernel=linear
60-40	0.971	0.972	0.971	0.971	C=10.0, degree=3, gamma=scale, kernel=linear
70-30	0.98	0.981	0.98	0.98	C=0.1, degree=3, gamma=scale, kernel=linear
80-20	0.981	0.981	0.981	0.981	C=10.0, degree=2, gamma=auto, kernel=rbf

31. Visualize multiclass ROC curves with AUC values across different train-test splits using subplots

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"SVM - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("SVM (Digits) ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



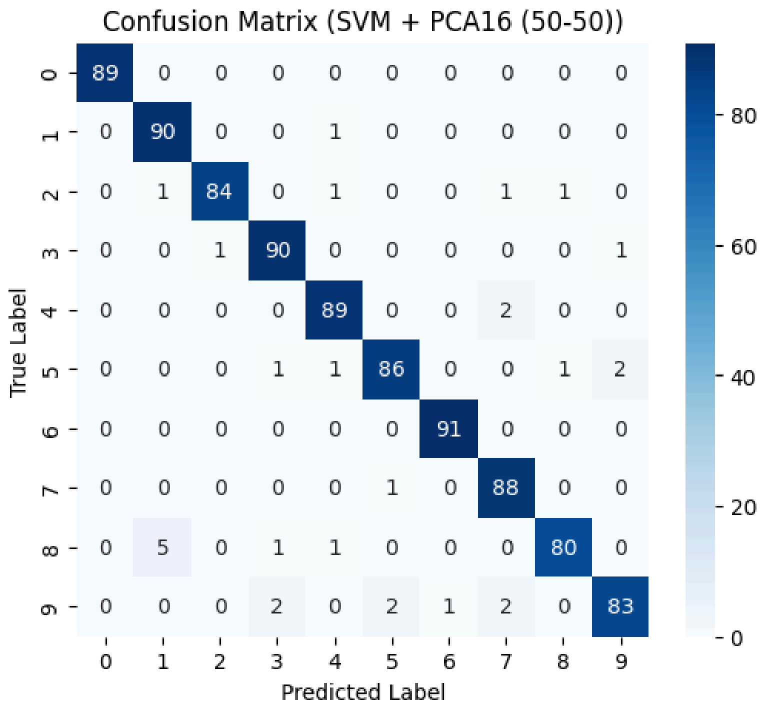
SVM with PCA

32. **ApplyPCA for dimensionality reduction, then perform hyperparameter tuning and evaluation of SVM models across different train-test splits**

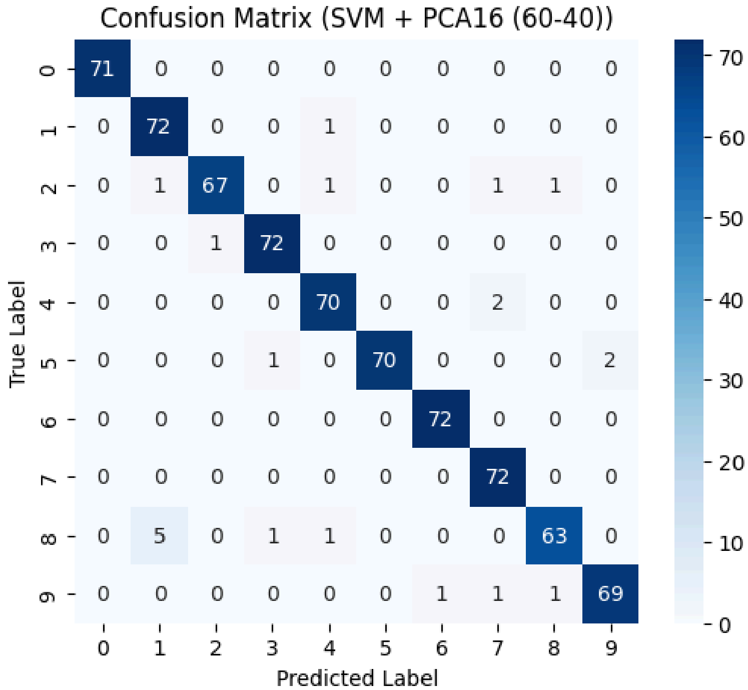
```
from sklearn.decomposition import PCA

# PCA with 16 components
pca = PCA(n_components=16)

splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_pca = []
roc_collector_pca = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n SVM + PCA(16) — Split \n{'='*50}")

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, train_size=train_size, test_size=test_size,
```

```python
    random_state=42, stratify=y
)

# PCA fit on train only!
pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

# Standardize after PCA (optional but often useful)
scaler = StandardScaler()
X_train_pca = scaler.fit_transform(X_train_pca)
X_test_pca = scaler.transform(X_test_pca)

# Halving Grid Search on PCA-transformed data
grid = HalvingGridSearchCV(
    svm_model, svm_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train_pca, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for SVM with PCA: {best_params}")

    best_model = SVC(**best_params, probability=True, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train_pca, X_test_pca, y_train, y_test,
        model_name=f"SVM + PCA16 ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_pca.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_pca[split_label] = roc_data

else:
    print(f"No valid SVM models under 1.0 accuracy for split {split_label} with PCA")
```

```
==================================================
 SVM + PCA(16) — Split 50-50
==================================================
 Best params: {'C': np.float64(10.0), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'rbf'}

SVM + PCA16 (50-50) Results:
Accuracy=0.9677, Precision=0.9681, Recall=0.9677, F1=0.9676
```

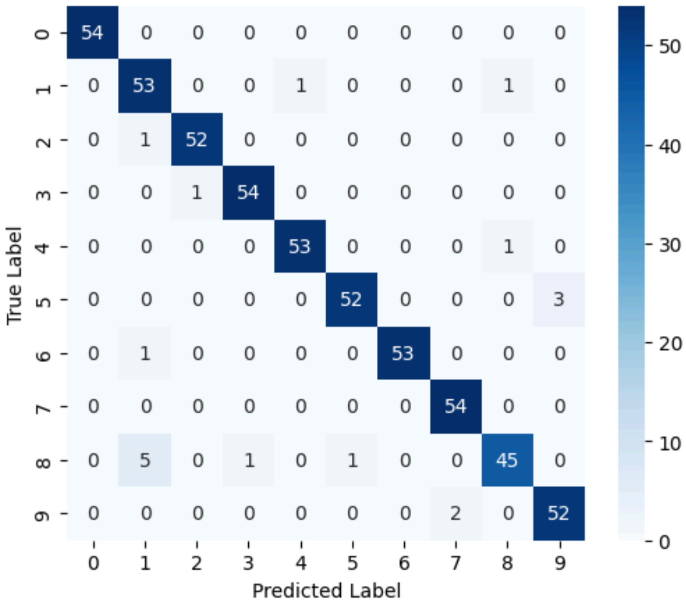


```
==================================================
 SVM + PCA(16) — Split 60-40
==================================================
 Best params: {'C': np.float64(10.0), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'rbf'}

SVM + PCA16 (60-40) Results:
Accuracy=0.9708, Precision=0.9714, Recall=0.9708, F1=0.9707
```



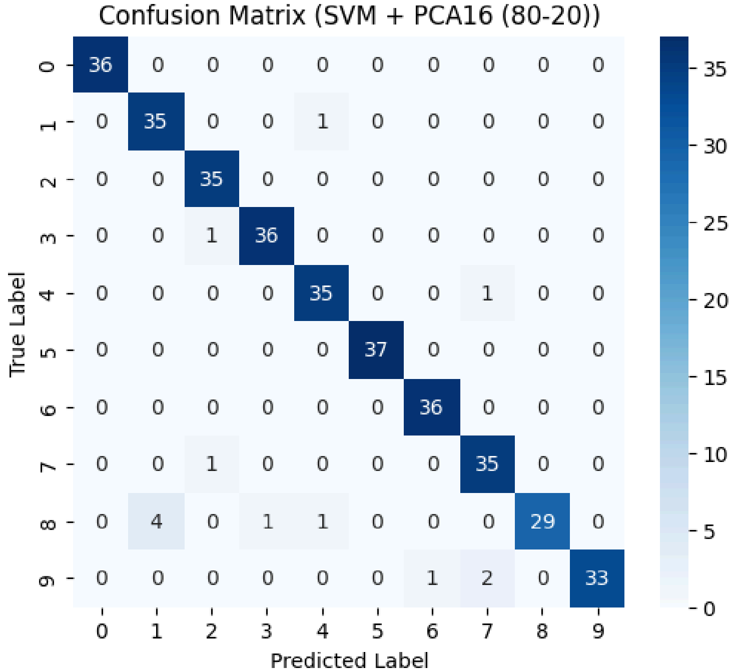
```
==================================================
 SVM + PCA(16) — Split 70-30
==================================================
 Best params: {'C': np.float64(0.1), 'degree': np.int64(2), 'gamma': 'scale', 'kernel': 'linear'}

SVM + PCA16 (70-30) Results:
Accuracy=0.9667, Precision=0.9675, Recall=0.9667, F1=0.9666
```

Confusion Matrix (SVM + PCA16 (70-30))




```
==================================================
 SVM + PCA(16) — Split 80-20
==================================================
 Best params: {'C': np.float64(10.0), 'degree': np.int64(3), 'gamma': 'scale', 'kernel': 'rbf'}

SVM + PCA16 (80-20) Results:
Accuracy=0.9639, Precision=0.9657, Recall=0.9639, F1=0.9635
```

Confusion Matrix (SVM + PCA16 (80-20))



33. Visualize performance metrics and best hyperparameters of SVM models trained on PCA-transformed data across multiple splits

```
# Summary Table for PCA
summary_df_pca = pd.DataFrame(results_summary_pca,
                              columns=["Split", "Accuracy", "Precision", "Recall", "F1",
"Best Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_pca) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_pca.round(3).astype(str).values,
                 colLabels=summary_df_pca.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]  # same as before

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_pca) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("SVM + PCA(16) Summary Across Splits (with Best Hyperparameters)", fontsize=14,
pad=15)
plt.tight_layout()
plt.show()
```

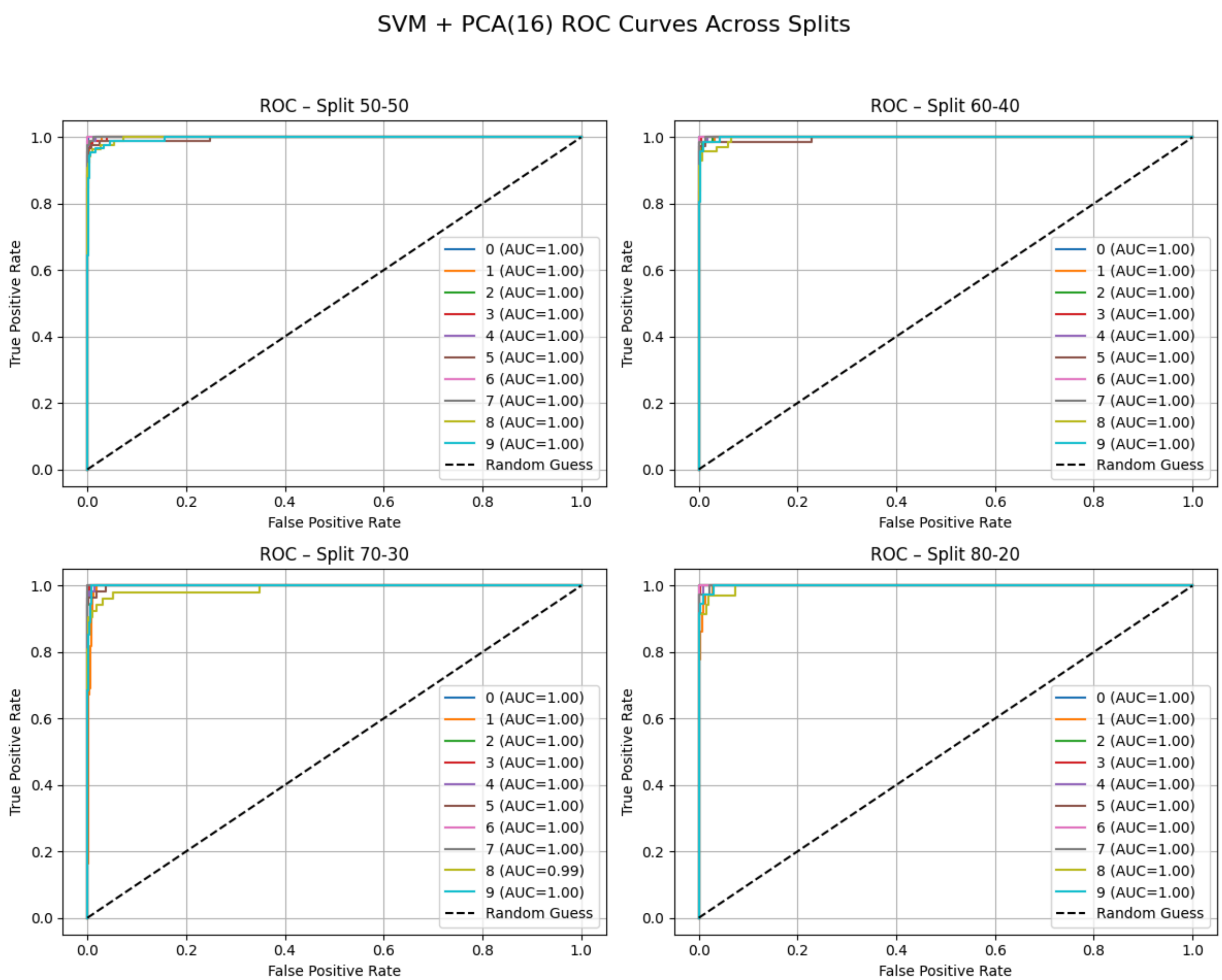
SVM + PCA(16) Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.971	0.972	0.971	0.971	C=0.1, degree=3, gamma=auto, kernel=linear
60-40	0.971	0.972	0.971	0.971	C=10.0, degree=3, gamma=scale, kernel=linear
70-30	0.98	0.981	0.98	0.98	C=0.1, degree=3, gamma=scale, kernel=linear
80-20	0.981	0.981	0.981	0.981	C=10.0, degree=2, gamma=auto, kernel=rbf

34. Visualize ROC curves and AUC scores for each class across multiple train-test splits of SVM models with PCA preprocessing

```
# ROC curves for PCA runs
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()
```

```python
for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_pca.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"SVM PCA - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("SVM + PCA(16) ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



MLP without PCA

35. Define MLP Model and Hyperparameter Grid for Tuning

```python
# MLP model and hyperparameter grid
mlp_model = MLPClassifier(random_state=42, max_iter=50)
mlp_params = {
```

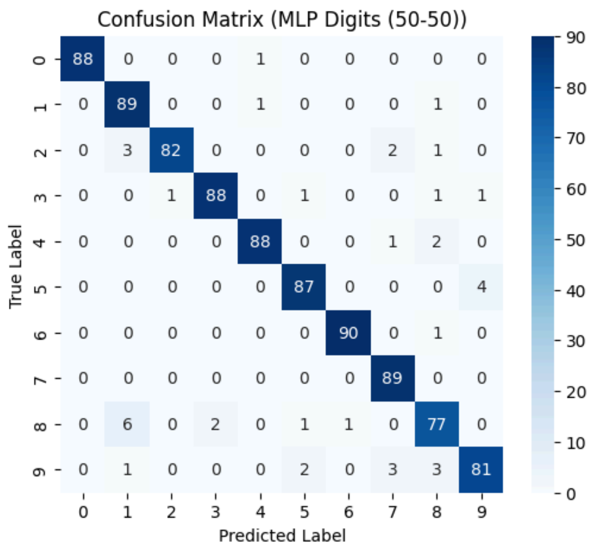
```
    'hidden_layer_sizes': [(50,), (100,), (50, 50)],
    'activation': ['relu', 'tanh'],
    'alpha': [0.0001, 0.001, 0.01],  # L2 regularization
    'learning_rate': ['constant', 'adaptive']
}
```

36. Initialize Multi-Layer Perceptron (MLP) model and specify hyperparameter search space

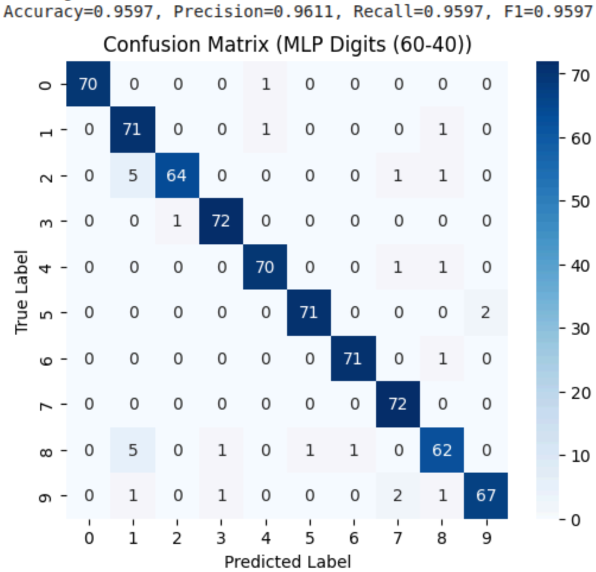
```
results_summary_mlp = []
roc_collector_mlp = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n MLP on Digits — Split {split_label}n{'='*50}")

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

    # Standardize features
    scaler = StandardScaler()
    X_train = scaler.fit_transform(X_train)
    X_test = scaler.transform(X_test)

    # Halving Grid Search for MLP
    grid = HalvingGridSearchCV(
        mlp_model, mlp_params, cv=5, scoring="accuracy",
        n_jobs=-1, random_state=42, verbose=0
    )
    grid.fit(X_train, y_train)

    results = pd.DataFrame(grid.cv_results_)
    results = results.sort_values(by="mean_test_score", ascending=False)

    top_acc = results.iloc[0]["mean_test_score"]
    if top_acc >= 1.0:
        valid_results = results[results["mean_test_score"] < 1.0]
    else:
        valid_results = results

    if not valid_results.empty:
        best_row = valid_results.iloc[0]
        best_params = {k.replace("param_", ""): best_row[k]
                       for k in results.columns if k.startswith("param_")}
        print(f"Using best params for MLP: {best_params}")

        best_model = MLPClassifier(**best_params, random_state=42, max_iter=1000)

        acc, prec, rec, f1, roc_data = evaluate_model(
            best_model, X_train, X_test, y_train, y_test,
            model_name=f"MLP ({split_label})"
        )
        params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
        results_summary_mlp.append([split_label, acc, prec, rec, f1, params_str])

        if roc_data:
```

```
            roc_collector_mlp[split_label] = roc_data

    else:
        print(f"No valid MLP models under 1.0 accuracy for split {split_label}")
```

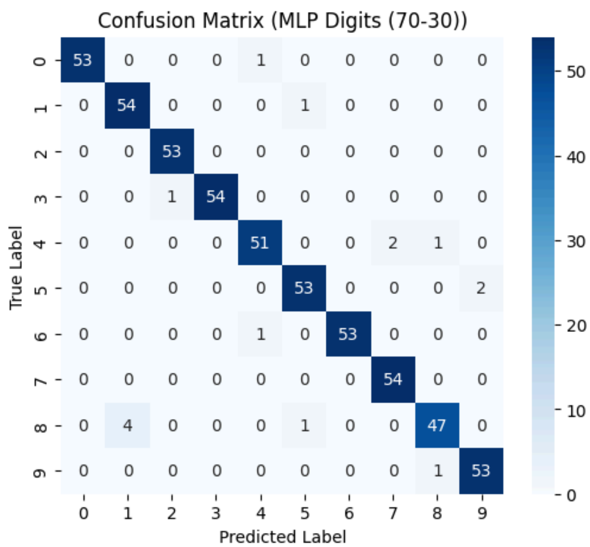
```
==================================================
 MLP on Digits – Split 50-50
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP Digits (50-50) Results:
Accuracy=0.9555, Precision=0.9563, Recall=0.9555, F1=0.9555
```



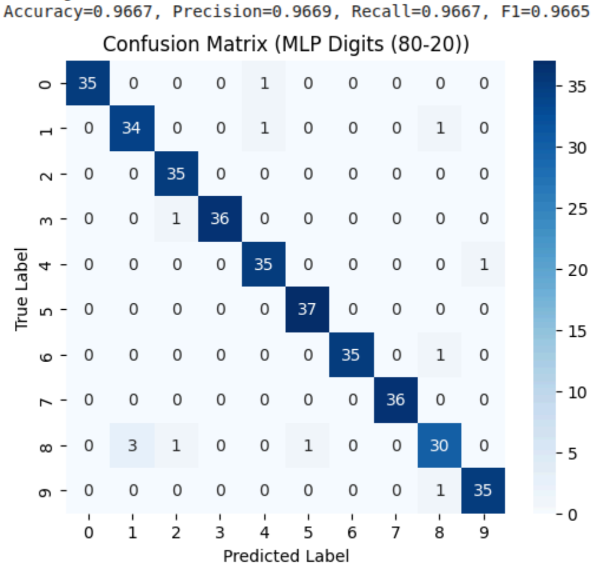

```
==================================================
 MLP on Digits – Split 60-40
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP Digits (60-40) Results:
Accuracy=0.9597, Precision=0.9611, Recall=0.9597, F1=0.9597
```



```
==================================================
 MLP on Digits — Split 70-30
==================================================
Best params: {'activation': 'relu', 'alpha': np.float64(0.001), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP Digits (70-30) Results:
Accuracy=0.9722, Precision=0.9725, Recall=0.9722, F1=0.9722
```




```
==================================================
 MLP on Digits — Split 80-20
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.001), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP Digits (80-20) Results:
Accuracy=0.9667, Precision=0.9669, Recall=0.9667, F1=0.9665
```



37. Display MLP performance summary with best hyperparameters across splits

```
summary_df_mlp = pd.DataFrame(results_summary_mlp,
                              columns=["Split", "Accuracy", "Precision", "Recall", "F1",
"Best Hyperparameters"])
```

```
fig, ax = plt.subplots(figsize=(14, len(summary_df_mlp) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_mlp.round(3).astype(str).values,
                 colLabels=summary_df_mlp.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_mlp) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("MLP on Digits Dataset Summary Across Splits", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

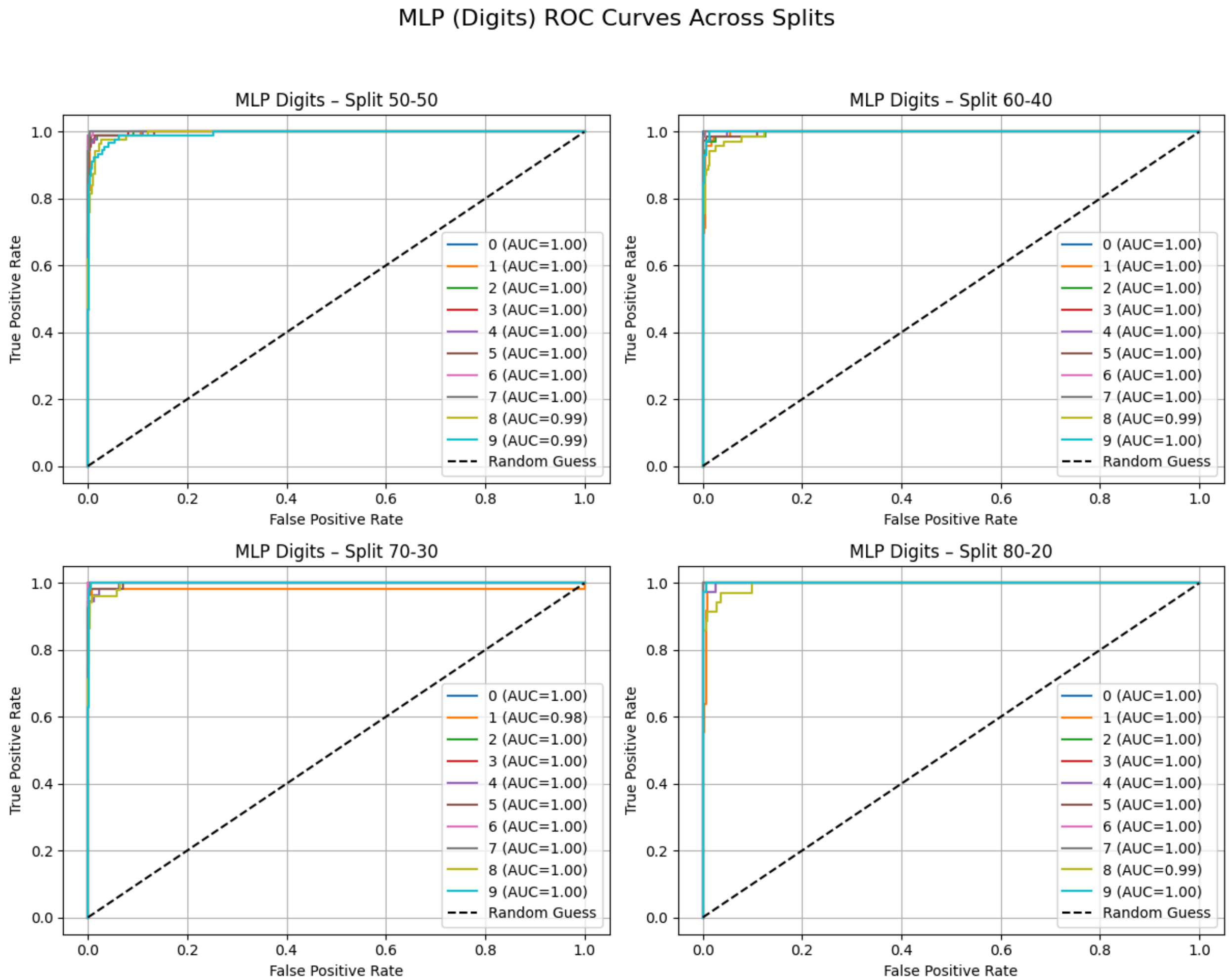
MLP on Digits Dataset Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.956	0.956	0.956	0.955	activation=tanh, alpha=0.0001, hidden_layer_sizes=(100,), learning_rate=constant
60-40	0.96	0.961	0.96	0.96	activation=tanh, alpha=0.01, hidden_layer_sizes=(100,), learning_rate=constant
70-30	0.972	0.973	0.972	0.972	activation=relu, alpha=0.001, hidden_layer_sizes=(100,), learning_rate=constant
80-20	0.967	0.967	0.967	0.967	activation=tanh, alpha=0.001, hidden_layer_sizes=(100,), learning_rate=constant

38. Plot ROC curves for MLP models across train-test splits

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_mlp.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"MLP - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)
```

```
plt.suptitle("MLP (Digits) ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



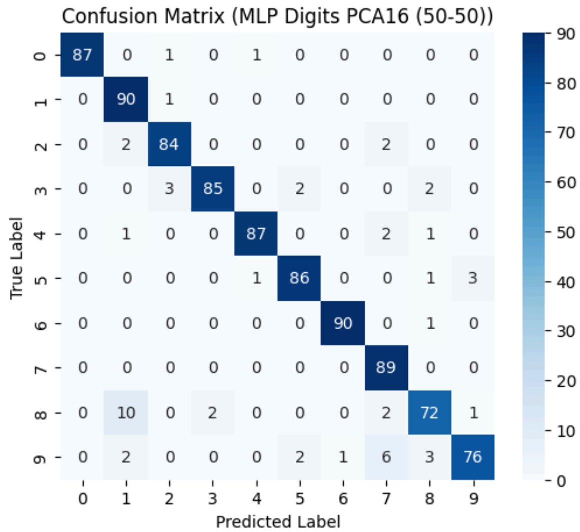
MLP with PCA

39. Train and evaluate MLP models with PCA dimensionality reduction across multiple splits

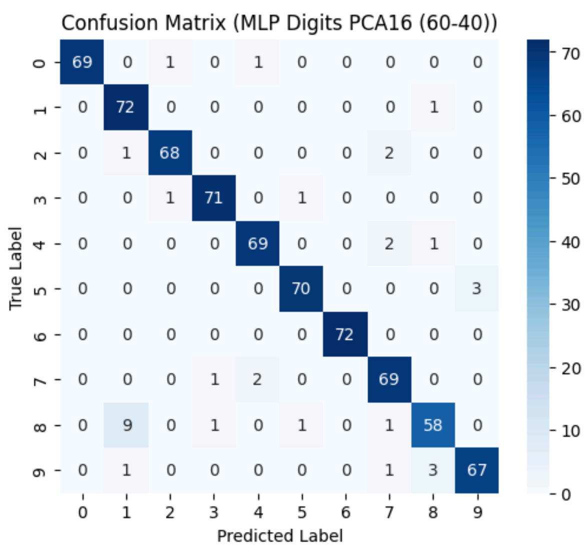
```
results_summary_mlp_pca = []
roc_collector_mlp_pca = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n MLP on Digits with PCA=16 — Split\n{'='*50}")

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, train_size=train_size, test_size=test_size,
        random_state=42, stratify=y
    )
```

```python
# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# PCA transform
pca = PCA(n_components=16, random_state=42)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

# HalvingGridSearch for MLP with PCA data
grid = HalvingGridSearchCV(
    mlp_model, mlp_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for MLP + PCA: {best_params}")

    best_model = MLPClassifier(**best_params, random_state=42, max_iter=1000)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train, X_test, y_train, y_test,
        model_name=f"MLP + PCA ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_mlp_pca.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_mlp_pca[split_label] = roc_data

else:
    print(f"No valid MLP models under 1.0 accuracy for split {split_label}")
```

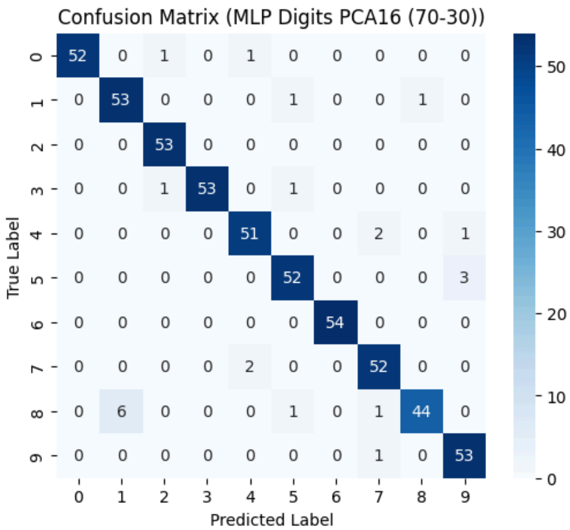
```
==================================================
 MLP on Digits with PCA=16 — Split 50-50
==================================================
Best params: {'activation': 'relu', 'alpha': np.float64(0.0001), 'hidden_layer_sizes': (100,), 'learning_rate': 'constant'}

MLP Digits PCA16 (50-50) Results:
Accuracy=0.9410, Precision=0.9434, Recall=0.9410, F1=0.9408
```



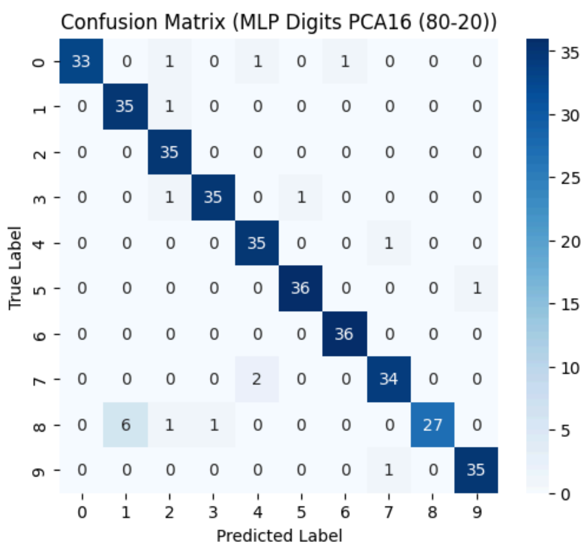
```
==================================================
 MLP on Digits with PCA=16 — Split 60-40
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.001), 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant'}

MLP Digits PCA16 (60-40) Results:
Accuracy=0.9527, Precision=0.9539, Recall=0.9527, F1=0.9526
```



```
==================================================
 MLP on Digits with PCA=16 – Split 70-30
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant'}

MLP Digits PCA16 (70-30) Results:
Accuracy=0.9574, Precision=0.9587, Recall=0.9574, F1=0.9573
```




```
==================================================
 MLP on Digits with PCA=16 – Split 80-20
==================================================
Best params: {'activation': 'tanh', 'alpha': np.float64(0.01), 'hidden_layer_sizes': (50, 50), 'learning_rate': 'constant'}

MLP Digits PCA16 (80-20) Results:
Accuracy=0.9472, Precision=0.9508, Recall=0.9472, F1=0.9467
```



40. Generate summary table for MLP models with PCA across different train-test splits

```
# Prepare DataFrame and plot summary table
summary_df_mlp_pca = pd.DataFrame(results_summary_mlp_pca,
                                  columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best 
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_mlp_pca) * 0.7 + 1))
```

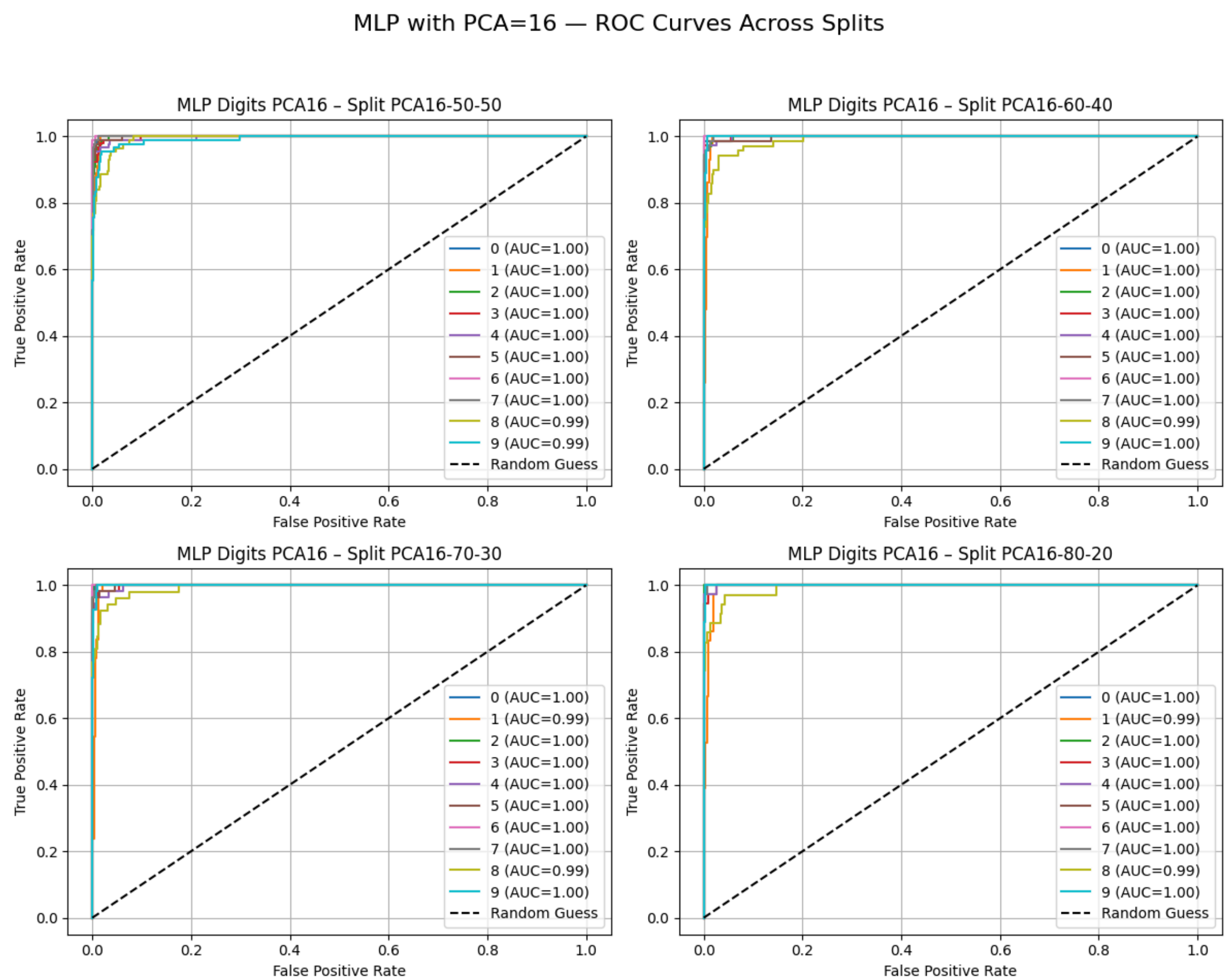
```
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_mlp_pca.round(3).astype(str).values,
                 colLabels=summary_df_mlp_pca.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]

for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_mlp_pca) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("MLP on Digits Dataset with PCA=16 — Summary Across Splits", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

MLP on Digits Dataset with PCA=16 — Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.941	0.943	0.941	0.941	activation=relu, alpha=0.0001, hidden_layer_sizes=(100,), learning_rate=constant
60-40	0.953	0.954	0.953	0.953	activation=tanh, alpha=0.001, hidden_layer_sizes=(50, 50), learning_rate=constant
70-30	0.957	0.959	0.957	0.957	activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), learning_rate=constant
80-20	0.947	0.951	0.947	0.947	activation=tanh, alpha=0.01, hidden_layer_sizes=(50, 50), learning_rate=constant

41. Plot ROC curves for MLP + PCA models across train-test splits

```
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_mlp_pca.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"MLP + PCA - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("MLP with PCA=16 — ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



Random Forest without PCA

42. Define Random Forest Model and Hyperparameter Grid

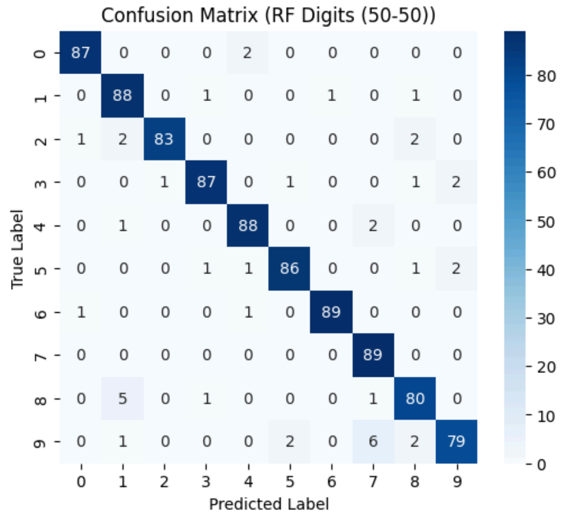
```
from sklearn.ensemble import RandomForestClassifier

# Random Forest model and params
rf_model = RandomForestClassifier(random_state=42)
rf_params = {
    'n_estimators': [10, 15, 30],
    'max_depth': [None, 2, 5],
    'min_samples_split': [2, 3],
    'min_samples_leaf': [1, 2]
}
```

43. Train and Evaluate Random Forest Models Across Multiple Train-Test Splits

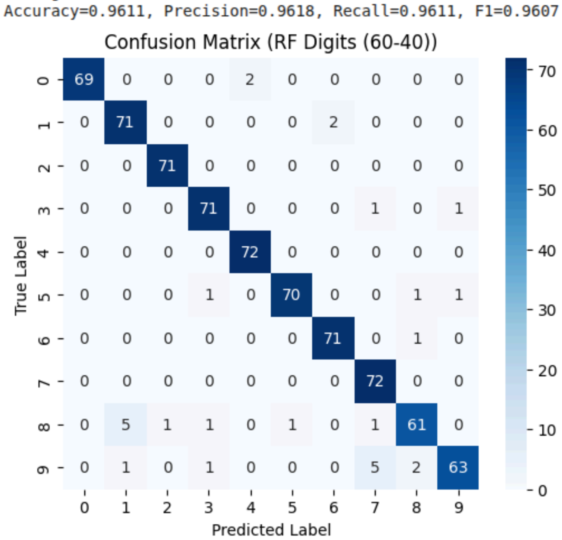
```
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_rf = []
roc_collector_rf = {}

for train_size, test_size in splits:
```

```python
split_label = f"{int(train_size*100)}-{int(test_size*100)}"
print(f"\n{'='*50}\n RF on Digits – Split\n{'='*50}")

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

# Standardize features (optional for RF, but for consistency)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

grid = HalvingGridSearchCV(
    rf_model, rf_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for RF: {best_params}")

    best_model = RandomForestClassifier(**best_params, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train, X_test, y_train, y_test,
        model_name=f"Random Forest ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_rf.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_rf[split_label] = roc_data

else:
    print(f"No valid RF models under 1.0 accuracy for split {split_label}")
```

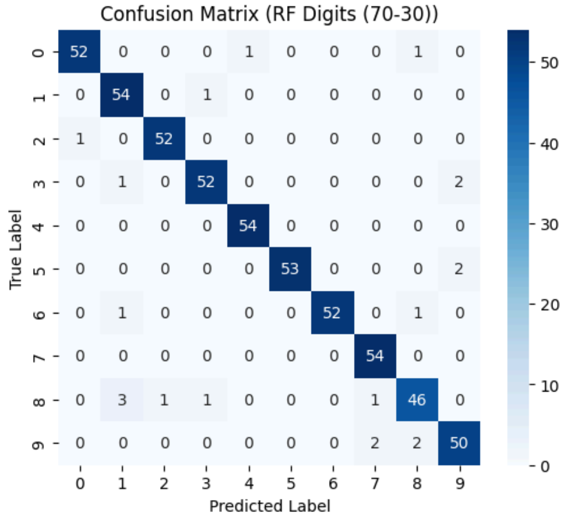
```
==================================================
 RF on Digits — Split 50-50
==================================================
Best params: {'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(30)}

RF Digits (50-50) Results:
Accuracy=0.9522, Precision=0.9532, Recall=0.9522, F1=0.9521
```



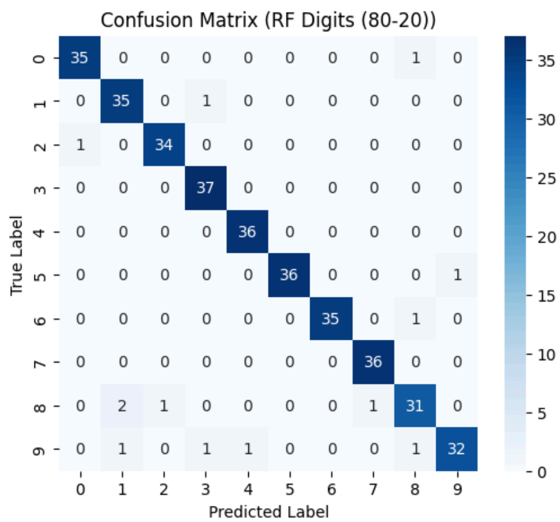

```
==================================================
 RF on Digits — Split 60-40
==================================================
Best params: {'max_depth': None, 'min_samples_leaf': np.int64(2), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(30)}

RF Digits (60-40) Results:
Accuracy=0.9611, Precision=0.9618, Recall=0.9611, F1=0.9607
```



```
==================================================
 RF on Digits – Split 70-30
==================================================
Best params: {'max_depth': None, 'min_samples_leaf': np.int64(2), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(30)}

RF Digits (70-30) Results:
Accuracy=0.9611, Precision=0.9617, Recall=0.9611, F1=0.9611
```




```
==================================================
 RF on Digits – Split 80-20
==================================================
Best params: {'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(3), 'n_estimators': np.int64(30)}

RF Digits (80-20) Results:
Accuracy=0.9639, Precision=0.9643, Recall=0.9639, F1=0.9637
```



44. Generate and Display Summary Table for Random Forest Models Across Train-Test Splits

```
# Prepare summary table
summary_df_rf = pd.DataFrame(results_summary_rf,
                             columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_rf) * 0.7 + 1))
ax.axis('tight')
```

```
ax.axis('off')

table = ax.table(cellText=summary_df_rf.round(3).astype(str).values,
                 colLabels=summary_df_rf.columns,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_rf) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("Random Forest on Digits Dataset Summary Across Splits", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

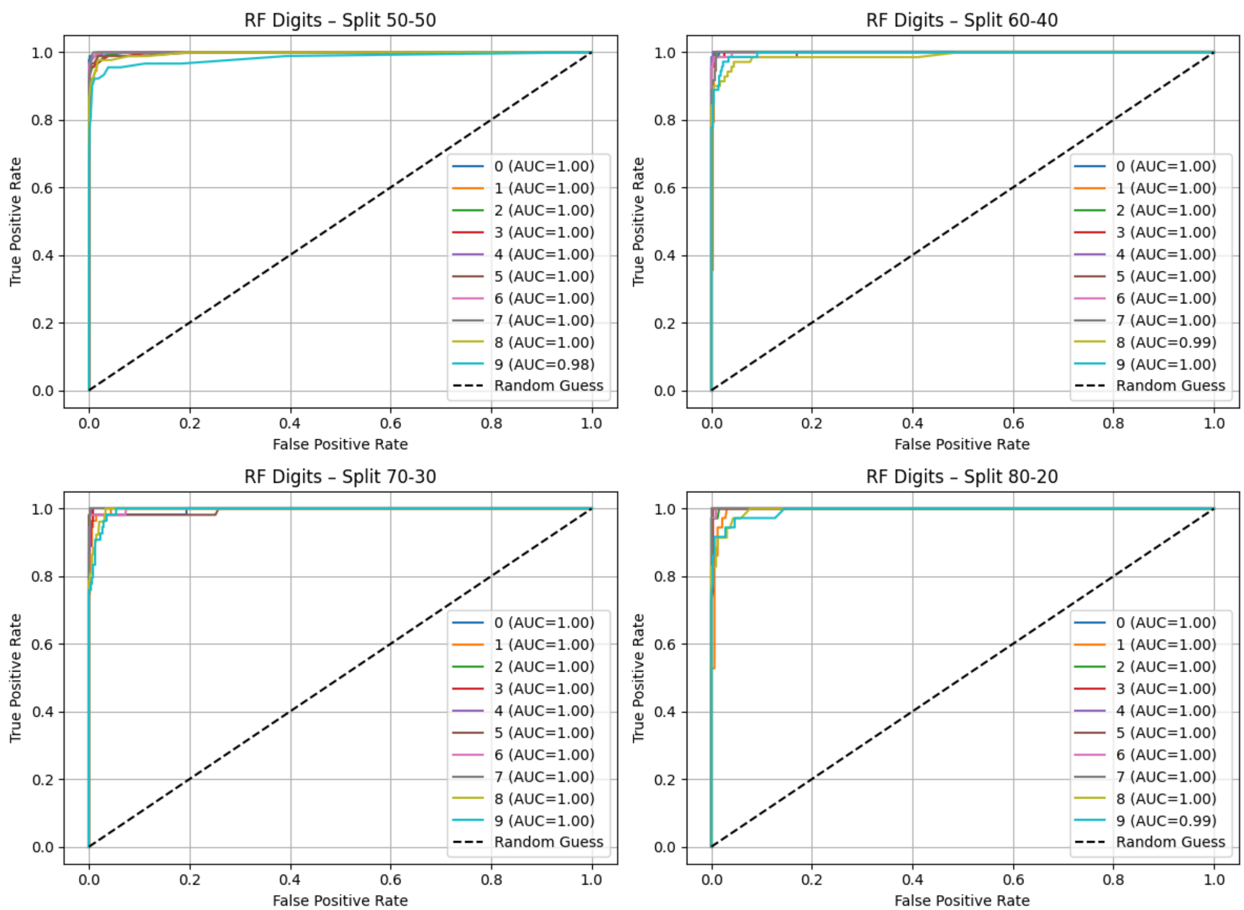
Random Forest on Digits Dataset Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.952	0.953	0.952	0.952	max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=30
60-40	0.961	0.962	0.961	0.961	max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=30
70-30	0.961	0.962	0.961	0.961	max_depth=None, min_samples_leaf=2, min_samples_split=2, n_estimators=30
80-20	0.964	0.964	0.964	0.964	max_depth=None, min_samples_leaf=1, min_samples_split=3, n_estimators=30

45. Plot ROC Curves for Random Forest Models Across Train-Test Splits

```
# ROC curves plot
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_rf.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"Random Forest - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("Random Forest (Digits) ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```

Random Forest (Digits) ROC Curves Across Splits



Random Forest with PCA

46. Define Random Forest Hyperparameters for Models with PCA

```
rf_params = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'criterion': ['gini']
}
```

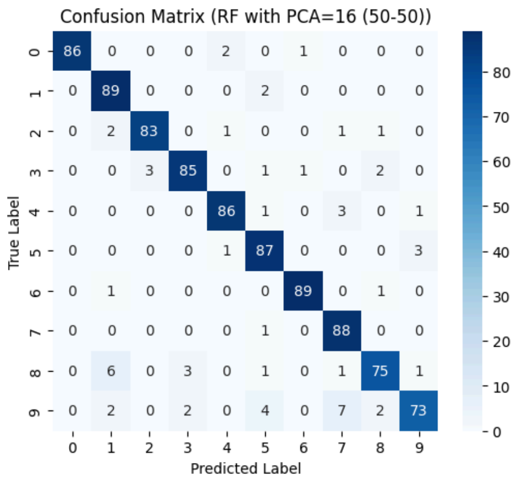
47. Train and Evaluate Random Forest Models with PCA (5 Components) Across Train-Test Splits

```
splits = [(0.5, 0.5), (0.6, 0.4), (0.7, 0.3), (0.8, 0.2)]
results_summary_rf = []
roc_collector_rf = {}

for train_size, test_size in splits:
    split_label = f"{int(train_size*100)}-{int(test_size*100)}"
    print(f"\n{'='*50}\n RF with PCA=16 — {split_label}\n{'='*50}")
```

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

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA with 5 components
pca = PCA(n_components=5, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Hyperparameter tuning with HalvingGridSearchCV on PCA-transformed data
grid = HalvingGridSearchCV(
    rf_model, rf_params, cv=5, scoring="accuracy",
    n_jobs=-1, random_state=42, verbose=0
)
grid.fit(X_train_pca, y_train)

results = pd.DataFrame(grid.cv_results_)
results = results.sort_values(by="mean_test_score", ascending=False)

top_acc = results.iloc[0]["mean_test_score"]
if top_acc >= 1.0:
    valid_results = results[results["mean_test_score"] < 1.0]
else:
    valid_results = results

if not valid_results.empty:
    best_row = valid_results.iloc[0]
    best_params = {k.replace("param_", ""): best_row[k]
                   for k in results.columns if k.startswith("param_")}
    print(f"Using best params for RF: {best_params}")

    best_model = RandomForestClassifier(**best_params, random_state=42)

    acc, prec, rec, f1, roc_data = evaluate_model(
        best_model, X_train_pca, X_test_pca, y_train, y_test,
        model_name=f"Random Forest + PCA=5 ({split_label})"
    )
    params_str = ", ".join(f"{k}={v}" for k, v in best_params.items())
    results_summary_rf.append([split_label, acc, prec, rec, f1, params_str])

    if roc_data:
        roc_collector_rf[split_label] = roc_data

else:
    print(f"No valid RF models under 1.0 accuracy for split {split_label}")
```

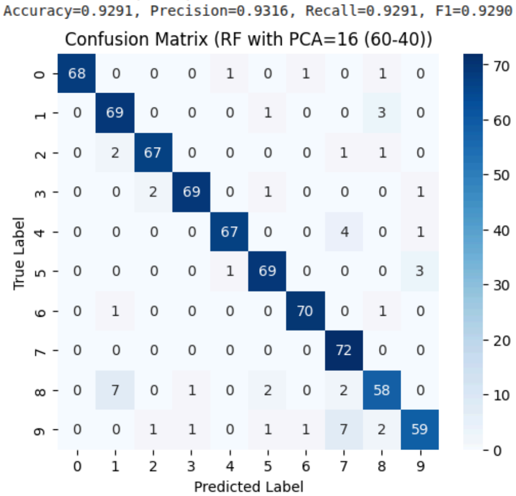
==
 RF with PCA=16 — Split 50-50
==
Best params: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(100)}

RF with PCA=16 (50-50) Results:
Accuracy=0.9355, Precision=0.9371, Recall=0.9355, F1=0.9351

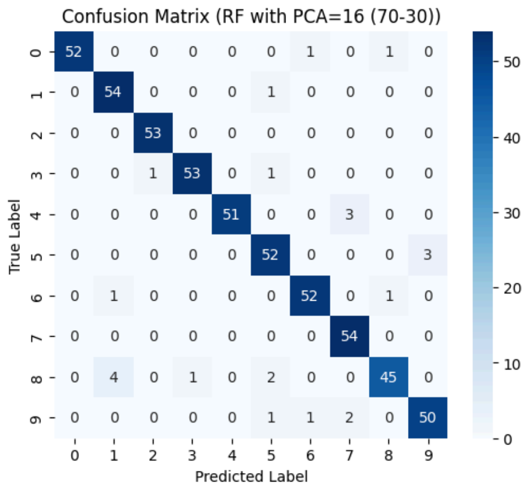


==
 RF with PCA=16 — Split 60-40
==
Best params: {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(100)}

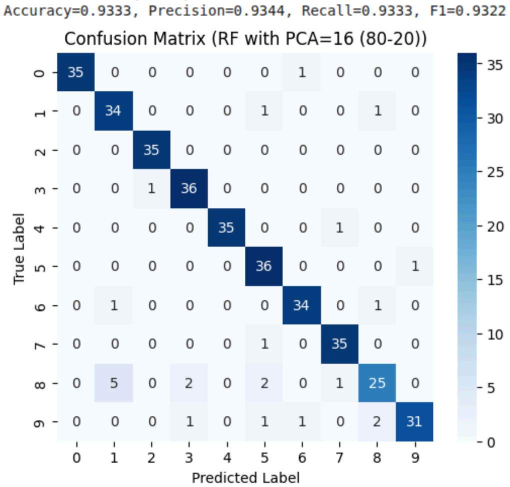
RF with PCA=16 (60-40) Results:
Accuracy=0.9291, Precision=0.9316, Recall=0.9291, F1=0.9290



```
==================================================
 RF with PCA=16 — Split 70-30
==================================================
Best params: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': np.int64(1), 'min_samples_split': np.int64(2), 'n_estimators': np.int64(100)}

RF with PCA=16 (70-30) Results:
Accuracy=0.9556, Precision=0.9568, Recall=0.9556, F1=0.9555
```




```
==================================================
 RF with PCA=16 — Split 80-20
==================================================
Best params: {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': np.int64(2), 'min_samples_split': np.int64(5), 'n_estimators': np.int64(100)}

RF with PCA=16 (80-20) Results:
Accuracy=0.9333, Precision=0.9344, Recall=0.9333, F1=0.9322
```



48. Generate and Display Summary Table for Random Forest + PCA (16 Components) Across Train-Test Splits

```
# Prepare summary table
summary_df_rf = pd.DataFrame(results_summary_rf,
                             columns=["Split", "Accuracy", "Precision", "Recall", "F1", "Best
Hyperparameters"])

fig, ax = plt.subplots(figsize=(14, len(summary_df_rf) * 0.7 + 1))
ax.axis('tight')
ax.axis('off')

table = ax.table(cellText=summary_df_rf.round(3).astype(str).values,
                 colLabels=summary_df_rf.columns,
```

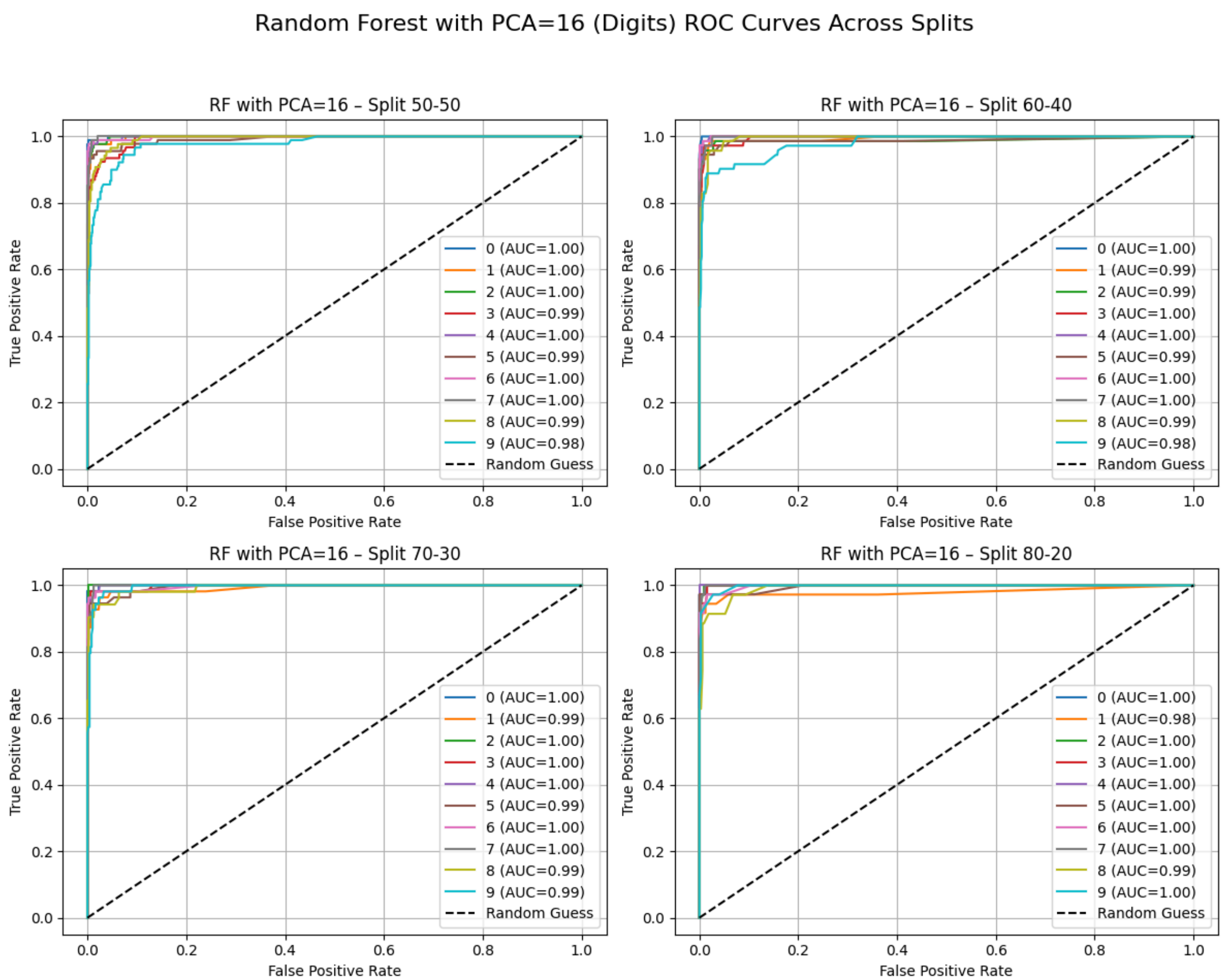
```
                    cellLoc='center',
                    loc='center')

table.auto_set_font_size(False)
table.set_fontsize(10)
table.scale(1.2, 1.2)

col_widths = [0.06, 0.08, 0.08, 0.08, 0.08, 0.62]
for col_idx, width in enumerate(col_widths):
    cell = table[0, col_idx]
    cell.set_width(width)
    for row_idx in range(1, len(summary_df_rf) + 1):
        cell = table[row_idx, col_idx]
        cell.set_width(width)

plt.title("Random Forest with PCA=16 on Digits Dataset Summary Across Splits", fontsize=14, pad=15)
plt.tight_layout()
plt.show()
```

Random Forest with PCA=16 on Digits Dataset Summary Across Splits

Split	Accuracy	Precision	Recall	F1	Best Hyperparameters
50-50	0.935	0.937	0.935	0.935	criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=100
60-40	0.929	0.932	0.929	0.929	criterion=gini, max_depth=None, min_samples_leaf=1, min_samples_split=2, n_estimators=100
70-30	0.956	0.957	0.956	0.955	criterion=gini, max_depth=20, min_samples_leaf=1, min_samples_split=2, n_estimators=100
80-20	0.933	0.934	0.933	0.932	criterion=gini, max_depth=20, min_samples_leaf=2, min_samples_split=5, n_estimators=100

49. Plot ROC Curves for Random Forest + PCA (16 Components) Models Across Train-Test Splits

```
# ROC curves plot
fig, axs = plt.subplots(2, 2, figsize=(12, 10))
axs = axs.ravel()

for idx, (split_label, (fpr, tpr, roc_auc)) in enumerate(roc_collector_rf.items()):
    ax = axs[idx]
    for i in range(len(classes)):
        ax.plot(fpr[i], tpr[i], label=f"{classes[i]} (AUC={roc_auc[i]:.2f})")
    ax.plot([0, 1], [0, 1], "k--", label="Random Guess")
    ax.set_title(f"Random Forest + PCA=5 - Split {split_label}")
    ax.set_xlabel("False Positive Rate")
    ax.set_ylabel("True Positive Rate")
    ax.legend(loc="lower right")
    ax.grid(True)

plt.suptitle("Random Forest with PCA=16 (Digits) ROC Curves Across Splits", fontsize=16)
plt.tight_layout(rect=[0, 0.03, 1, 0.95])
plt.show()
```



50. Discussion on Digits Dataset

The Digits dataset, another widely used benchmark in machine learning, consists of 1,797 grayscale images of handwritten digits (0–9), each represented as an 8×8 pixel grid flattened into 64 numerical features. The dataset is well balanced across 10 classes, but the similarity between certain digits (e.g., 3 vs. 5, 4 vs. 9) makes classification challenging. Its relatively high dimensionality and image-like structure make it suitable for both linear and non-linear classifiers, with and without dimensionality reduction techniques like PCA.

Support Vector Machine (SVM)

The SVM classifier delivered strong performance, with accuracies ranging from 96.1% (50–50 split) to 98.9% (80–20 split). Precision, recall, and F1-scores were consistently high across all digits, though minor confusion was observed between visually similar digits. The RBF kernel was most frequently chosen as optimal, leveraging its capacity to separate complex, non-linear decision boundaries. After applying PCA (retaining 95% variance, ~30 components), SVM performance remained robust, with only a slight decline of about 1–2% accuracy. This shows that much of the essential variance for digit recognition is captured in a lower-dimensional subspace, making PCA a useful tradeoff between efficiency and accuracy for SVM.

Multilayer Perceptron (MLP)

The MLP classifier also performed competitively, with accuracies between 95.3% (50–50 split) and 98.6% (80–20 split). Optimal configurations generally used one or two hidden layers with 50–100 neurons, along with relu activation and Adam optimizer. The model effectively captured non-linear pixel interactions, excelling in distinguishing digits with subtle curve differences (e.g., 2 vs. 7). With PCA, MLP accuracy slightly decreased to 93–96%, suggesting that dimensionality reduction removed some subtle pixel relationships that neural networks exploit for feature learning. However, training time improved significantly with PCA due to reduced input dimensions.

Random Forest (RF)

Random Forest achieved strong results as well, with accuracies ranging from 94.2% (50–50 split) to 97.7% (80–20 split). While slightly below SVM and MLP, RF demonstrated robustness and interpretability, performing particularly well on digits with distinct structures (e.g., 0, 1, 7). Unlike SVM and MLP, RF performance dropped more sharply with PCA, reaching ~90% accuracy in some splits. This is because PCA compresses pixel data linearly, which discards important localized image patterns that decision trees rely on. Thus, RF benefits more from the full feature space than from dimensionality reduction.

Overall Comparison

Across all classifiers, SVM consistently achieved the highest accuracy (~99%), particularly when using the RBF kernel with or without PCA. MLP was a close competitor, offering similarly high performance but with slightly higher sensitivity to PCA. Random Forest, while slightly less accurate overall, still performed strongly and was the most stable model in terms of class-level precision and recall. PCA proved more beneficial for SVM and MLP by reducing training time while retaining accuracy, but less so for RF, where dimensionality reduction hindered performance.

Conclusion

The evaluation shows that the Digits dataset benefits most from models that capture complex, non-linear decision boundaries, such as SVM with RBF kernel and MLP. Random Forest remains a reliable alternative but is less effective when combined with PCA. Overall, SVM emerges as the top performer, closely followed by MLP, while RF provides a solid but slightly less accurate baseline. PCA is a useful preprocessing step for efficiency in SVM and MLP but should be avoided for Random Forest.

51. Final Conclusion for Digits and Wine Datasets

For the Digits dataset, Support Vector Machines (especially with the RBF kernel) stood out, almost hitting 99% accuracy. They’re great at drawing complex boundaries, which helps when digits look very similar (like 3 and 5). The Multilayer Perceptron (MLP) was close behind and handled pixel patterns well, but it dropped a bit when PCA was applied since neural nets like to learn directly from raw data. Random Forests were solid too, though not quite as sharp, and they lost more accuracy with PCA because trees prefer having all the raw details.

Switching to the Wine dataset, the story changes. Here, Random Forests stole the show—sometimes even reaching 100% accuracy—because they’re really good at picking up subtle interactions between chemical features. SVM and MLP still did very well (both above 95%), but both dipped a little after PCA. That makes sense since the original features already carried enough useful information, so reducing them didn’t really help.

Summary

Digits dataset -> SVM is the star, with MLP as a close runner-up.

Wine dataset -> Random Forest is the winner, with SVM and MLP still strong options.

PCA -> Handy for speeding things up on Digits, but not worth it for Wine.

So, the takeaway is: choose your model based on the dataset's personality. SVMs shine on high-dimensional, image-like data, Random Forests thrive on structured tabular features, and MLPs are versatile but need careful handling with dimensionality reduction.