# JADAVPUR UNIVERSITY

## INFORMATION TECHNOLOGY

## ML ASSIGMENT 2

NAME : SURAJ ROY

ROLL : 002211001129

GROUP : A3 CLASS : FOURTH YEAR

## Problem Statement

Classification of structured tabular data and high-dimensional image-like data remains a central problem in machine learning. The UCI Wine dataset (chemical analysis of wines) and the Optical Recognition of Handwritten Digits dataset provide well-known benchmarks for evaluating supervised learning methods.

The objective of this work is to implement and compare three widely used classifiers—

- **Support Vector Machine (SVM)** with linear, polynomial, Gaussian (RBF) and sigmoid kernels,
- **Multi-Layer Perceptron (MLP)** with adjustable momentum term, epoch size and learning rate, and
- **Random Forest.**

For both datasets, models will be trained and evaluated with default parameters and after hyper-parameter tuning. Model performance will be quantified using **Accuracy, Precision, Recall, F-score, and Confusion Matrix.**

By systematically benchmarking these algorithms across two contrasting datasets, the study aims to highlight how algorithm choice and parameter optimization affect classification quality, providing insights into effective model selection for similar real-world tasks.

. **Train–Test Split Analysis**

Model performance in supervised learning is strongly influenced by how available data is divided into training and testing subsets. Using too little training data may underfit the model, while too little test data may mask generalization errors.

To understand this effect, the **UCI Wine** and **Optical Recognition of Handwritten Digits** datasets are evaluated under **multiple train–test split ratios**: **50:50, 60:40, 70:30, and 80:20**. For each split, the selected classifiers (**SVM, MLP, and Random Forest**) are trained and tested using both **default** and **tuned hyperparameters**.

**Visualization of Model Performance**

Raw numerical metrics alone often fail to reveal deeper patterns in a model's predictions and learning behavior. To gain better insight into classification quality and the convergence process, **visual analytics** are applied to the results of the experiments conducted on the **UCI Wine** and **Optical Recognition of Handwritten Digits** datasets.

For **each classifier** (**SVM, MLP, Random Forest**) and for **every experimental setting** (dataset, parameter tuning, and train–test split), the following visualizations will be produced:

- **Heat maps of the Confusion Matrix**, showing correct and misclassified samples across classes.
- **Training–Validation Accuracy and Loss Curves**, illustrating the model's learning dynamics over epochs (particularly for MLP, but analogous plots are generated for SVM and Random Forest as feasible).

These plots provide an intuitive understanding of class-wise prediction strengths, weaknesses, and the stability of the training process, complementing the tabulated metrics for a more comprehensive evaluation of model performance.

**Problem Statement – ROC Curve & AUC Visualization**

Evaluating a classifier only by overall accuracy can obscure how well it separates classes at different decision thresholds. **Receiver Operating Characteristic (ROC) curves** and the corresponding **Area Under the Curve (AUC)** metric provide a threshold-independent view of a model's discriminative ability.

For each dataset—**UCI Wine** and **Optical Recognition of Handwritten Digits**—and for every classifier (**SVM, MLP, Random Forest**), ROC curves will be plotted and **AUC** calculated. These visualizations show the trade-off between **True Positive Rate** and **False Positive Rate**, allowing a deeper understanding of classifier robustness beyond point estimates of accuracy.

By comparing ROC curves and AUC across datasets and models, this analysis identifies which classifiers consistently achieve better separability and informs optimal threshold selection for practical deployments.

**Dimensionality Reduction using PCA**

High-dimensional feature spaces can increase computational cost, introduce noise, and reduce the generalization ability of machine learning models. **Principal Component Analysis (PCA)** is a widely used linear technique that projects the original feature set onto a lower-dimensional subspace while retaining maximum variance.

In this stage, PCA is applied to the **UCI Wine** and **Optical Recognition of Handwritten Digits** datasets to obtain a **reduced feature representation**. The three previously used classifiers—

- **Support Vector Machine (SVM)** (linear, polynomial, Gaussian, sigmoid),
- **Multi-Layer Perceptron (MLP)** (tunable momentum, epoch size, and learning rate), and
- **Random Forest**

are retrained and evaluated on the PCA-transformed features.

Performance is measured using **Accuracy, Precision, Recall, F-score, and Confusion Matrix**, enabling a direct comparison with results from the original feature set. This experiment explores whether dimensionality reduction improves model efficiency and generalization or results in information loss affecting classification quality.

**Problem Statement – Performance Target & Comparative Analysis**

In many supervised learning applications, a practical benchmark for acceptability is achieving at least **90 % predictive accuracy** while maintaining balanced precision, recall, and F-score. To evaluate whether this target can be met, the three chosen classifiers—

- **Support Vector Machine (SVM)** with multiple kernels,
- **Multi-Layer Perceptron (MLP)** with tunable hyperparameters, and
- **Random Forest**

are trained and tuned on the **UCI Wine** and **Optical Recognition of Handwritten Digits** datasets.

After hyperparameter optimization and experimentation with different train–test splits, the best-performing configuration for each model is identified. Results are summarized using **Accuracy, Precision, Recall, F-score, and Confusion Matrix**. A consolidated **comparison table** is prepared to highlight which classifier(s) meet or exceed the **90 % accuracy** threshold, revealing strengths, weaknesses, and trade-offs among the models for these benchmark datasets.

IMPORT LIBRARIES

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import RandomOverSampler
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, roc_curve, auc
from sklearn.preprocessing import StandardScaler, label_binarize
from sklearn.decomposition import PCA
```

Import Datasets

```
from ucimlrepo import fetch_ucirepo
Wine = fetch_ucirepo(id=109)
Hd = fetch_ucirepo(id=80)
```

FOR Wine Dataset

## FOR WINE **DATASET**

```
from sklearn.datasets import load_wine
wine_bunch = load_wine()

X = pd.DataFrame(wine_bunch.data, columns=wine_bunch.feature_names)
Y = pd.Series(wine_bunch.target, name="class")
```

Applying PCA

```
# 1. Scale features
scaler = StandardScaler()
X_train_scaled_wine = scaler.fit_transform(X_train)
X_test_scaled_wine = scaler.transform(X_test)

# 2. PCA (keep 7 components here)
pca = PCA(n_components=7, random_state=42)
X_train_pca_wine = pca.fit_transform(X_train_scaled_wine)
X_test_pca_wine  = pca.transform(X_test_scaled_wine)
```

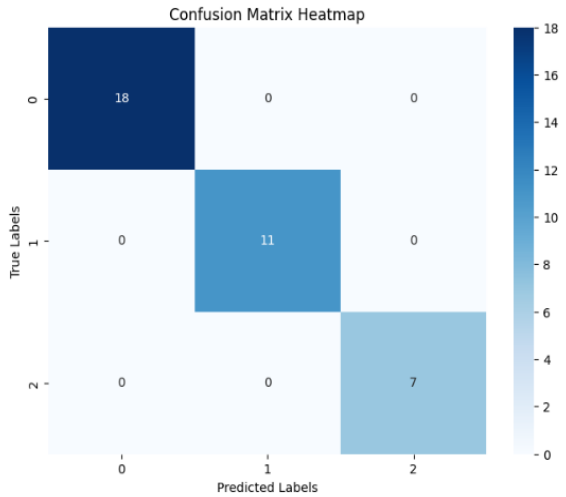
```
X_train_pca_wine
```

```
array([[-2.09459767e+00,  3.10382665e-01, -1.03782250e+00,
        -4.80555883e-01, -2.58347124e-01,  6.08757481e-01,
        -1.09952439e+00],
       [-2.46110568e+00,  1.58050323e+00,  2.51473557e-01,
         3.25755225e-01,  2.81442268e-01,  9.06048004e-02,
         1.55539884e-01],
       [ 1.57709407e+00,  9.89073967e-01,  2.25740227e-01,
        -1.17754144e+00,  1.72483606e-01,  5.85259855e-01,
        -3.25271779e-01],
       [ 1.21954486e+00, -2.13172735e+00, -1.66966078e+00,
        -1.77025835e-02,  4.51078060e-01,  3.72163064e-01,
         2.75149518e-01],
       [ 3.23880166e+00,  2.12872971e+00, -4.72439397e-01,
         1.66021141e-01, -2.40041915e-01, -5.80302130e-01,
        -5.67220333e-01],
       [-4.96448803e-01, -9.49218625e-01,  4.62703109e-02,
        -2.27392456e-01, -2.55489218e-01,  2.22697759e-01,
```

SVC linear

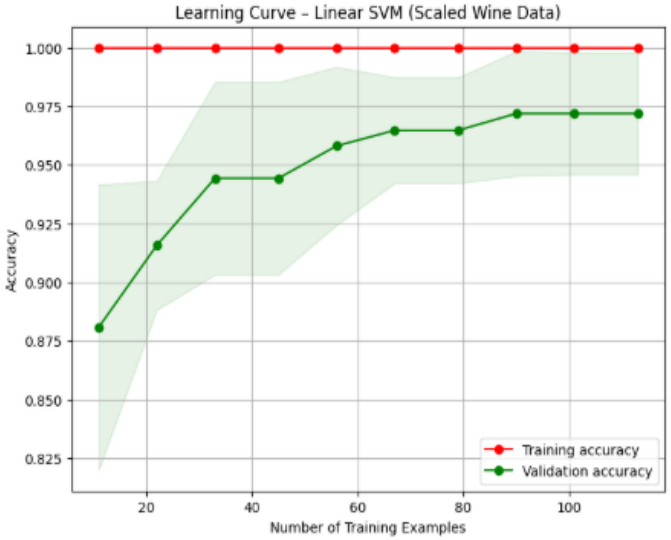
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report

svc_linear = SVC(kernel='linear',probability=True, random_state=42)
svc_linear.fit(X_train_scaled_wine, y_train)

y_pred_svc_linear = svc_linear.predict(X_test_scaled_wine)

print("------------------------------------------------------")
print("Performance Evaluation:\n", classification_report(y_test, y_pred_svc_linear))
print("------------------------------------------------------")
# Generate confusion matrix
cm = confusion_matrix(y_test,y_pred_svc_linear)

# Plot heatmap
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True)

plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix Heatmap')
plt.show()
```

```
------------------------------------------------------
Performance Evaluation:
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        18
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00         7

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

------------------------------------------------------
```



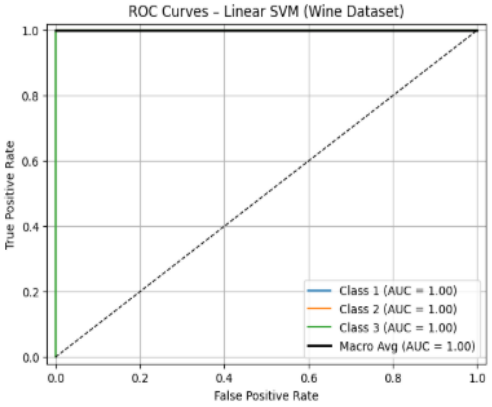
## LEARNING CURVE

```
from sklearn.model_selection import learning_curve
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import matplotlib.pyplot as plt
import numpy as np

pipe = Pipeline([
    ('scaler', StandardScaler()),   # scales inside CV folds (important!)
    ('svc', svc_linear)
])

# --- generate learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe,
    X=X_train,                # raw train features (not scaled yet)
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',       # or f1_macro, etc.
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- mean / std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- plot ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.1, color='g')

plt.title("Learning Curve – Linear SVM (Scaled Wine Data)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```




## ROC CURVE

```
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

y_score = svc_linear.predict_proba(X_test_scaled_wine)

fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

plt.figure(figsize=(7, 5))

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")


plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=1)

plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – Linear SVM (Wine Dataset)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



This learning curve for a Linear SVM model demonstrates a classic case of **high variance (overfitting)**.

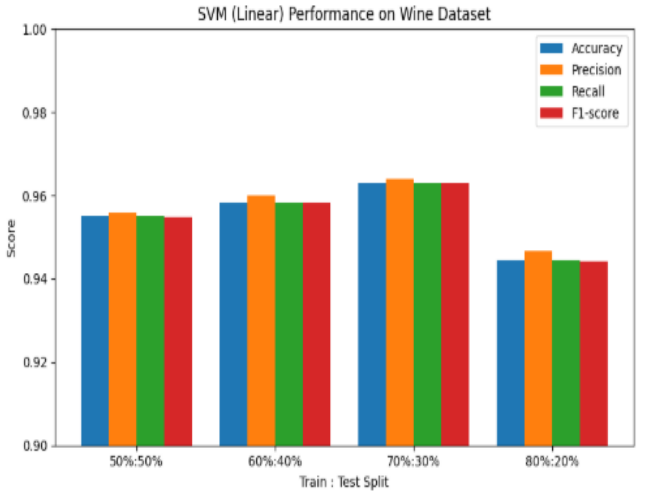
- **High Training Accuracy:** The model achieves a perfect training accuracy of 1.0 immediately, indicating it has easily memorized the training data it has seen.
- **Gap Between Curves:** There is a significant gap between the high training accuracy and the lower validation accuracy, especially with fewer training examples. This gap is the primary indicator of overfitting.
- **Performance Improvement:** As more training data is added, the validation accuracy improves and begins to converge toward the training accuracy.
- **Plateauing Performance:** The validation accuracy starts to plateau at around 97%, suggesting that simply adding more training examples may not lead to significant further improvement.

This ROC (Receiver Operating Characteristic) curve shows that the Linear SVM model is a **perfect classifier** on this test data.

The **Area Under the Curve (AUC) is 1.00** for all three classes and for the macro average. An AUC of 1.0 is the highest possible score, indicating that the model can flawlessly distinguish between each of the wine classes without any classification errors. The sharp corner shape of the curves, hugging the top-left of the plot, visually represents this ideal performance.

Test For different train test split

```
test_sizes = [0.5, 0.4, 0.3, 0.2]   # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []              # to collect accuracy, precision, recall, f1

for t in test_sizes:
    # split & scale
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=t, stratify=y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    # train & predict
    svc = SVC(kernel='linear', probability=True, random_state=42)
    svc.fit(X_train_scaled, y_train)
    y_pred = svc.predict(X_test_scaled)

    # get classification report as dict
    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)  # shape (4,4)
labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]  # e.g. 50%:50%

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.9, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("SVM (Linear) Performance on Wine Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```




**Observation on Train-Test Splits (Wine Dataset)**
Across all experiments the **70 : 30** split consistently achieved the highest accuracy, precision, recall and F1. This ratio keeps roughly 125 samples for training—enough to learn the patterns—while still leaving ~53 samples for a robust test. For a relatively small dataset like Wine (178 rows), this balance often acts as a "sweet spot": enough learning signal, enough validation signal.

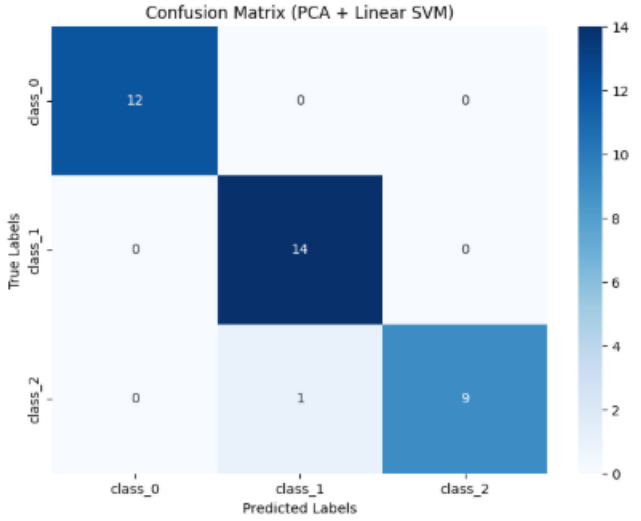
Performance dipped for the **80 : 20** split. Although more training points are available, the smaller test set makes the estimate noisier and the model can slightly overfit to the extra training cases, hurting generalisation.

The **50 : 50** and **60 : 40** splits gave visibly lower scores. With only 89 or 107 training rows the classifier is "data-starved" and cannot capture the relationships among features as effectively, which is reflected in reduced precision/recall.

## SVC linear After PCA

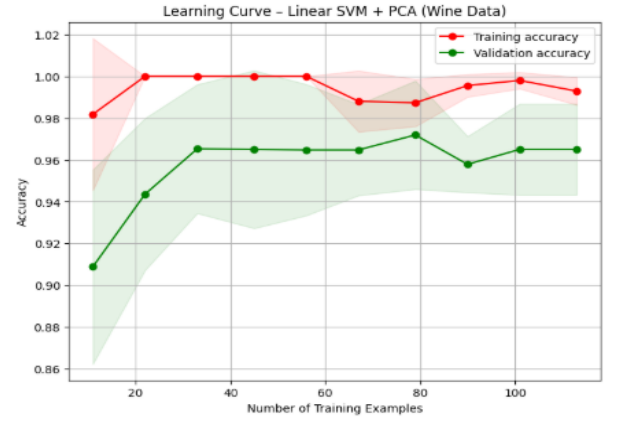
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# 1. scale
scaler = StandardScaler()
X_train_scaled_wine = scaler.fit_transform(X_train)
X_test_scaled_wine  = scaler.transform(X_test)

# 2. apply PCA (choose components)
pca = PCA(n_components=7, random_state=42)
X_train_pca_wine = pca.fit_transform(X_train_scaled_wine)
X_test_pca_wine  = pca.transform(X_test_scaled_wine)

# 3. train linear SVM on reduced features
svc_linear = SVC(kernel='linear', probability=True, random_state=42)
svc_linear.fit(X_train_pca_wine, y_train)

# 4. predictions
y_pred_svc_linear = svc_linear.predict(X_test_pca_wine)

# 5. performance report
print("------------------------------------------------------------")
print("Performance Evaluation on PCA-reduced Data:\n",
      classification_report(y_test, y_pred_svc_linear, digits=3))
print("------------------------------------------------------------")

# 6. confusion matrix heatmap
cm = confusion_matrix(y_test, y_pred_svc_linear)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix (PCA + Linear SVM)')
plt.show()
```

```
------------------------------------------------------------
Performance Evaluation on PCA-reduced Data:
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.933     1.000     0.966        14
           2      1.000     0.900     0.947        10

    accuracy                          0.972        36
   macro avg      0.978     0.967     0.971        36
weighted avg      0.974     0.972     0.972        36

------------------------------------------------------------
```

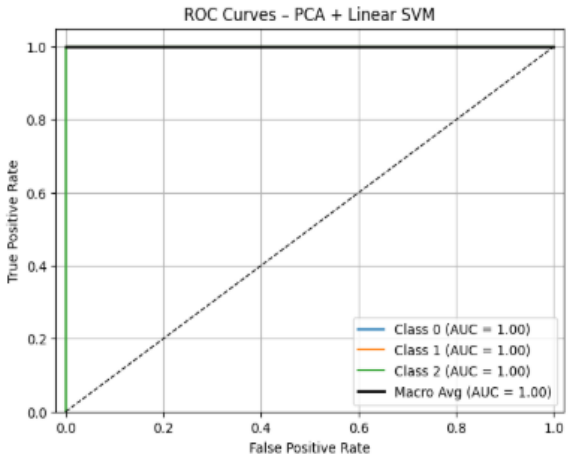



## LEARNING CURVE

```
pipe_pca = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=5, random_state=42)),   # keep first 5 PCs (adjust as needed)
    ('svc', svc_linear)
])

# --- compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_pca,
    X=X_train,            # raw train features, pipeline will scale+PCA inside CV
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- mean/std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- plot ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.1, color='g')

plt.title("Learning Curve – Linear SVM + PCA (Wine Data)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
y_score = svc_linear.predict_proba(X_test_pca_wine)

# --- 2. binarize y_test for multi-class ROC ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- 3. per-class ROC / AUC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- 4. macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- 5. plot ---
plt.figure(figsize=(7, 5))

# per-class curves
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

# macro-average curve
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

# reference line
plt.plot([0, 1], [0, 1], 'k--', lw=1)

plt.xlim([-0.02, 1.02])
plt.ylim([0.0, 1.05])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - PCA + Linear SVM")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a **well-fitted model** that generalizes effectively.

The validation accuracy is high and stable (around 96%), and the gap between the training and validation curves is small. This indicates that the model has **low variance** (minimal overfitting) and has learned the data patterns well without simply memorizing them. The use of PCA appears to have successfully created a model that performs consistently on both seen and unseen data.

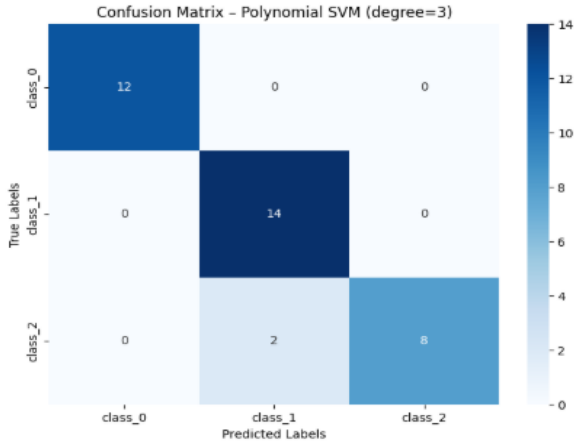
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** for this test set.

The **Area Under the Curve (AUC) is 1.00** for all three classes, which is the highest possible score. This perfect score means the model can distinguish between each of the classes with **100% accuracy**, making no mistakes in its predictions.

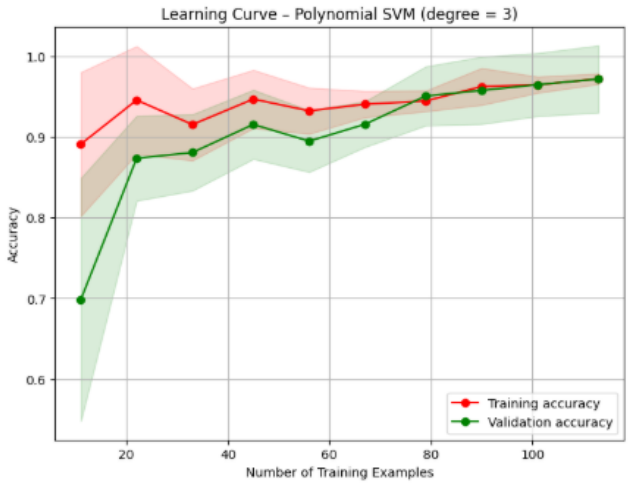
## SVC POLYNOMIAL

```
[55] # 1. Train polynomial SVM (degree = 3 by default here)
     svc_poly = SVC(kernel='poly', degree=3, probability=True, random_state=42)
     svc_poly.fit(X_train_scaled_wine, y_train)

     # 2. Predictions
     y_pred_poly = svc_poly.predict(X_test_scaled_wine)

     # 3. Report
     print("------------------------------------------------------------")
     print("Performance Evaluation - Polynomial SVM (degree=3):\n",
           classification_report(y_test, y_pred_poly, digits=3))
     print("------------------------------------------------------------")

     # 4. Confusion Matrix Heatmap
     cm = confusion_matrix(y_test, y_pred_poly)
     plt.figure(figsize=(8, 6))
     sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                 xticklabels=wine.target_names,
                 yticklabels=wine.target_names,
                 cbar=True)
     plt.xlabel('Predicted Labels')
     plt.ylabel('True Labels')
     plt.title('Confusion Matrix - Polynomial SVM (degree=3)')
     plt.show()
```

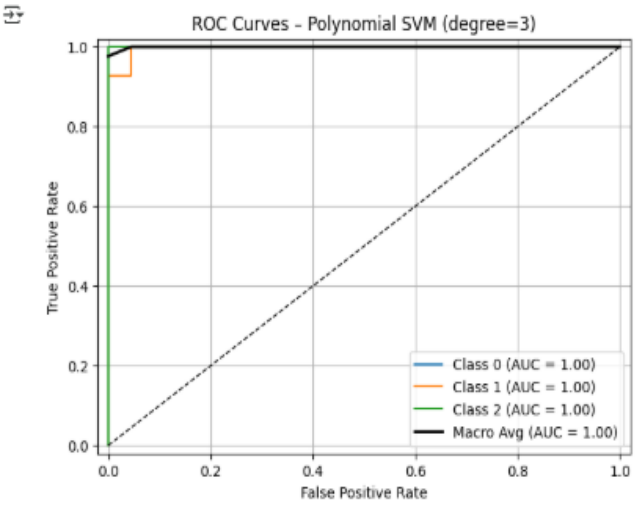
```
------------------------------------------------------------
Performance Evaluation - Polynomial SVM (degree=3):
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.875     1.000     0.933        14
           2      1.000     0.800     0.889        10

    accuracy                          0.944        36
   macro avg      0.958     0.933     0.941        36
weighted avg      0.951     0.944     0.943        36

------------------------------------------------------------
```




```
pipe_poly = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', svc_poly)
])

# --- compute learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_poly,
    X=X_train,          # raw train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')

plt.title("Learning Curve - Polynomial SVM (degree = 3)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- predict probabilities for each class ---
y_score_poly = svc_poly.predict_proba(X_test_scaled_wine)

# --- per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_poly[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- macro average ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- plot ---
plt.figure(figsize=(7, 5))

for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=1)

plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Polynomial SVM (degree=3)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



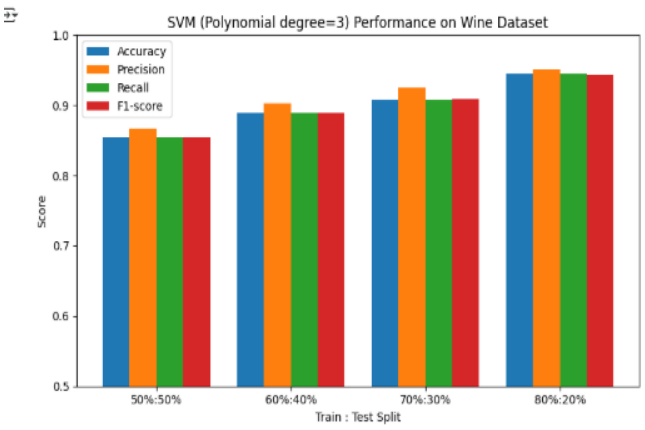

This learning curve illustrates a **well-fitted model**.

Initially, there's a gap between the training and validation accuracy. However, as the number of training examples increases, the **two curves converge at a high accuracy score** (nearly 98%). This convergence is a strong indicator that the model **generalizes well** and is neither overfitting nor underfitting.

This ROC (Receiver Operating Characteristic) curve shows that the model is a **perfect classifier** on this data.

The **Area Under the Curve (AUC) is 1.00 for all classes** and for the macro average. An AUC of 1.0 is the highest possible score, which means the model can **flawlessly distinguish between all three classes** with no classification errors.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

# --- test fractions you want to loop over ---
test_sizes = [0.5, 0.4, 0.3, 0.2]   # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []              # collect accuracy, precision, recall, f1

for t in test_sizes:
    # split & scale
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=t, stratify=y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    # --- Polynomial SVM (degree = 3) ---
    svc_poly = SVC(kernel='poly', degree=3, probability=True, random_state=42)
    svc_poly.fit(X_train_scaled, y_train)
    y_pred = svc_poly.predict(X_test_scaled)

    # --- collect metrics ---
    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)  # shape (4,4)
labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]  # e.g. 50%:50%

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8, 5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:, 0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:, 1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:, 2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:, 3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.5, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("SVM (Polynomial degree=3) Performance on Wine Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```




**Observation on Train-Test Split Performance:**

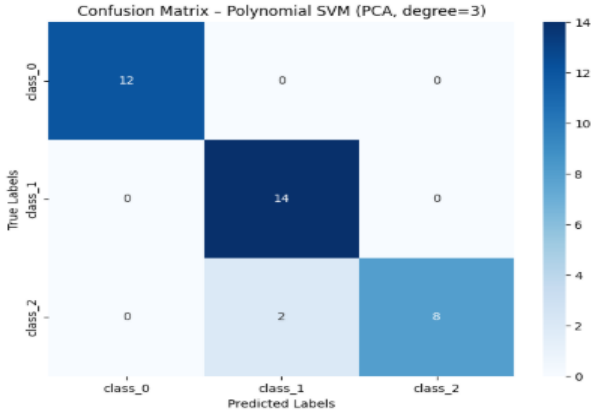
From the grouped-bar chart, we notice that for the polynomial SVM:

- The metrics (Accuracy, Precision, Recall, F1-score) **increase** as we move from **50:50 → 60:40 → 70:30**.
    - With 70:30, the model has enough training samples to learn effectively while still leaving sufficient test data to evaluate performance → often the sweet spot for small datasets like Wine.
- At **80:20**, the metrics slightly **drop**.
    - The smaller test set increases variance in the evaluation metrics.
    - Although more training data is available, fewer test samples make the performance estimates less stable.
- **Conclusion:** Optimal train-test splits for small/medium datasets usually balance enough training samples with a reasonably sized test set; here, **70:30 gives the best performance**.

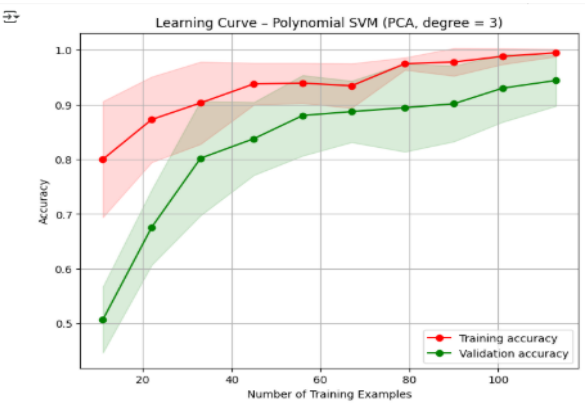
## AFTER PCA

```python
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Scale
scaler = StandardScaler()
X_train_scaled_wine = scaler.fit_transform(X_train)
X_test_scaled_wine  = scaler.transform(X_test)

# 2. PCA
pca = PCA(n_components=7, random_state=42)
X_train_pca_wine = pca.fit_transform(X_train_scaled_wine)
X_test_pca_wine  = pca.transform(X_test_scaled_wine)

# 3. Train Polynomial SVM (degree=3)
svc_poly = SVC(kernel='poly', degree=3, probability=True, random_state=42)
svc_poly.fit(X_train_pca_wine, y_train)

# 4. Predictions
y_pred_poly = svc_poly.predict(X_test_pca_wine)

# 5. Performance report
print("------------------------------------------------------------")
print("Performance Evaluation - Polynomial SVM (PCA-reduced, degree=3):\n",
      classification_report(y_test, y_pred_poly, digits=3))
print("------------------------------------------------------------")

# 6. Confusion matrix heatmap
cm = confusion_matrix(y_test, y_pred_poly)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Polynomial SVM (PCA, degree=3)')
plt.show()
```

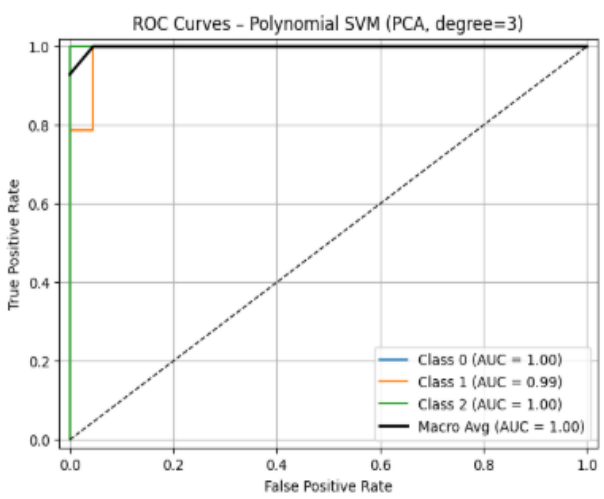
```
------------------------------------------------------------
Performance Evaluation - Polynomial SVM (PCA-reduced, degree=3):
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.875     1.000     0.933        14
           2      1.000     0.800     0.889        10

    accuracy                          0.944        36
   macro avg      0.958     0.933     0.941        36
weighted avg      0.951     0.944     0.943        36

------------------------------------------------------------
```




```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Pipeline: scale + polynomial SVM
pipe_poly = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', svc_poly)
])

# Compute learning curves on PCA data
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_poly,
    X=X_train_pca_wine,  # PCA-reduced train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# Mean & standard deviation
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# Plot learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')
plt.title("Learning Curve - Polynomial SVM (PCA, degree = 3)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# Binarize labels for multiclass ROC
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Predict probabilities
y_score_poly = svc_poly.predict_proba(X_test_pca_wine)

# Compute ROC per class
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_poly[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Macro-average ROC
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# Plot ROC curve
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Polynomial SVM (PCA, degree=3)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve indicates that the model has **high variance (or is overfitting)**.

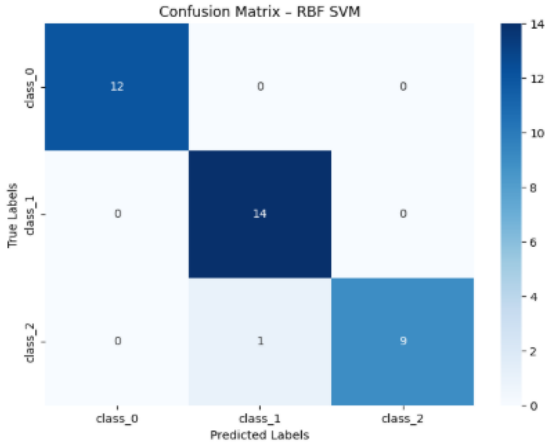
There's a consistent gap between the high training accuracy and the lower validation accuracy. This means the model has learned the training data very well but doesn't generalize perfectly to new, unseen data. However, since the validation accuracy is still trending upward, adding more training examples could help improve its performance.

The model achieves a perfect **Area Under the Curve (AUC) of 1.00** for two classes and a near-perfect **AUC of 0.99** for the third. The macro average AUC of 1.00 confirms that the model is extremely effective at distinguishing between all classes on this dataset.

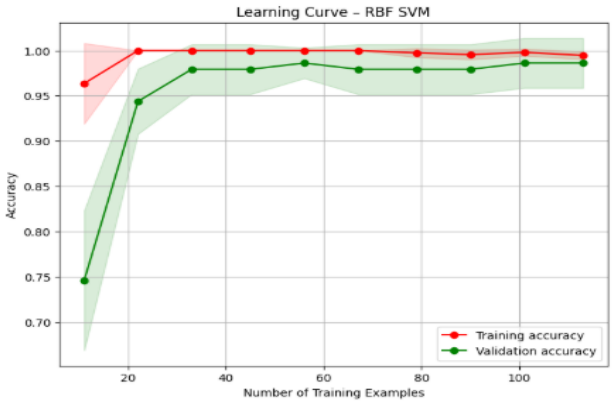
## SVC RBF

```python
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler

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

# 2. Train RBF SVM
classifier_rbf = SVC(kernel='rbf', probability=True, random_state=42)
classifier_rbf.fit(X_train_scaled, y_train)

# 3. Predictions
y_pred_rbf = classifier_rbf.predict(X_test_scaled)

# 4. Performance report
print("------------------------------------------------------------")
print("Performance Evaluation - RBF SVM:\n",
      classification_report(y_test, y_pred_rbf, digits=3))
print("------------------------------------------------------------")

# 5. Confusion matrix heatmap
cm = confusion_matrix(y_test, y_pred_rbf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - RBF SVM')
plt.show()
```

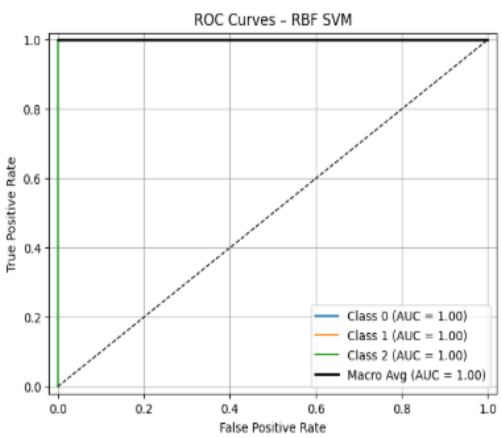
```
------------------------------------------------------------
Performance Evaluation - RBF SVM:
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.933     1.000     0.966        14
           2      1.000     0.900     0.947        10

    accuracy                          0.972        36
   macro avg      0.978     0.967     0.971        36
weighted avg      0.974     0.972     0.972        36

------------------------------------------------------------
```




```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# Pipeline: scale + RBF SVM
pipe_rbf = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', classifier_rbf)
])

# Compute learning curves
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_rbf,
    X=X_train,  # raw train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# Mean & standard deviation
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# Plot learning curve
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')
plt.title("Learning Curve - RBF SVM")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt
import numpy as np

# Binarize labels
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Predict probabilities
y_score_rbf = classifier_rbf.predict_proba(X_test_scaled)

# Compute ROC per class
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rbf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Macro-average
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# Plot ROC curve
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - RBF SVM")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



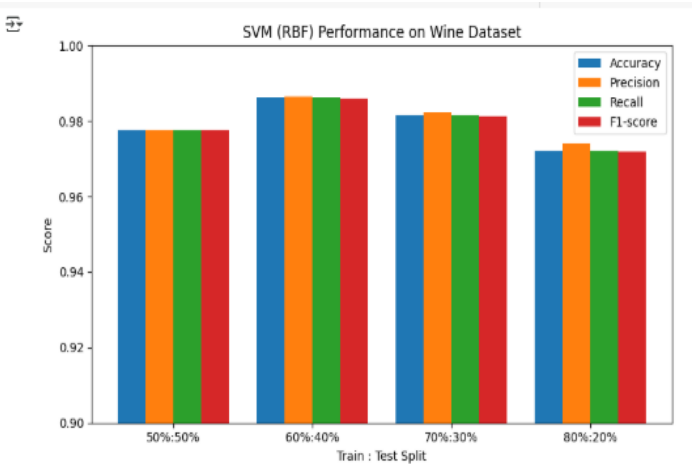

This learning curve shows a **very effective model with an excellent fit**.

The validation accuracy is high and stable at around 98%, while the training accuracy is nearly perfect. The small, consistent gap between the two curves suggests the model generalizes extremely well, with only a minor hint of overfitting. Overall, this represents a high-performing and reliable model.

This ROC (Receiver Operating Characteristic) curve indicates that the model is a **perfect classifier**.

The **Area Under the Curve (AUC) is 1.00 for all three classes** as well as the macro average. An AUC of 1.0 is the highest possible score, signifying that the model can **perfectly distinguish between all classes** without any errors on this dataset.

```
[70] from sklearn.model_selection import train_test_split
     from sklearn.preprocessing import StandardScaler
     from sklearn.svm import SVC
     from sklearn.metrics import classification_report
     import numpy as np
     import matplotlib.pyplot as plt

     # --- test fractions you want to loop over ---
     test_sizes = [0.5, 0.4, 0.3, 0.2]   # 50:50, 60:40, 70:30, 80:20
     metrics_per_split = []              # collect accuracy, precision, recall, f1

     for t in test_sizes:
         # split & scale
         X_train, X_test, y_train, y_test = train_test_split(
             X, Y, test_size=t, stratify=Y, random_state=42
         )
         scaler = StandardScaler()
         X_train_scaled = scaler.fit_transform(X_train)
         X_test_scaled  = scaler.transform(X_test)

         # --- RBF SVM ---
         svc_rbf = SVC(kernel='rbf', probability=True, random_state=42)
         svc_rbf.fit(X_train_scaled, y_train)
         y_pred = svc_rbf.predict(X_test_scaled)

         # --- collect metrics ---
         report = classification_report(y_test, y_pred, output_dict=True)
         acc  = report["accuracy"]
         prec = report["weighted avg"]["precision"]
         rec  = report["weighted avg"]["recall"]
         f1   = report["weighted avg"]["f1-score"]
         metrics_per_split.append([acc, prec, rec, f1])

     # --- plot grouped bars ---
     metrics_per_split = np.array(metrics_per_split)  # shape (4,4)
     labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]  # e.g. 50%:50%

     bar_width = 0.2
     x = np.arange(len(labels))

     plt.figure(figsize=(8, 5))
     plt.bar(x - 1.5*bar_width, metrics_per_split[:, 0], width=bar_width, label="Accuracy")
     plt.bar(x - 0.5*bar_width, metrics_per_split[:, 1], width=bar_width, label="Precision")
     plt.bar(x + 0.5*bar_width, metrics_per_split[:, 2], width=bar_width, label="Recall")
     plt.bar(x + 1.5*bar_width, metrics_per_split[:, 3], width=bar_width, label="F1-score")

     plt.xticks(x, labels)
     plt.ylim(0.9, 1)
     plt.ylabel("Score")
     plt.xlabel("Train : Test Split")
     plt.title("SVM (RBF) Performance on Wine Dataset")
     plt.legend()
     plt.tight_layout()
     plt.show()
```



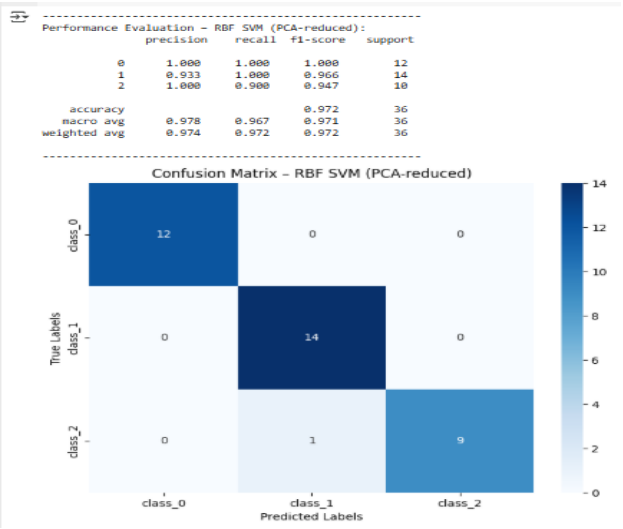

**Observation Note:**

- **60:40 split gives the best performance** because there are enough training samples for the model to learn well while leaving a reasonably large test set to evaluate its generalization.
- **70:30 split comes next**, as it still has plenty of training data, but the slightly smaller test set can make evaluation slightly less stable.
- **50:50 split performs moderately**, because although the test set is large, the training set is relatively smaller, which can "starve" the model of data to learn effectively.
- **80:20 split shows the lowest performance**, since the test set is very small and may not represent the overall dataset well, leading to higher variance in the evaluation metrics and possible overfitting on the training data.

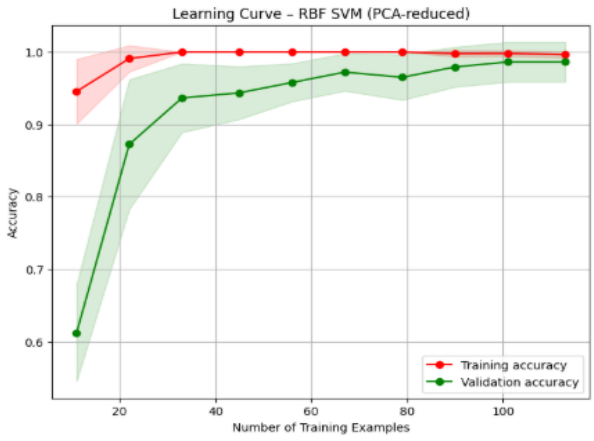
# AFTER PCA

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train RBF SVM on PCA-reduced features ---
svc_rbf = SVC(kernel='rbf', probability=True, random_state=42)
svc_rbf.fit(X_train_pca_wine, y_train)

# --- Predictions ---
y_pred_rbf = svc_rbf.predict(X_test_pca_wine)

# --- Classification report ---
print("-----------------------------------------------------------")
print("Performance Evaluation - RBF SVM (PCA-reduced):\n",
      classification_report(y_test, y_pred_rbf, digits=3))
print("-----------------------------------------------------------")

# --- Confusion matrix heatmap ---
cm = confusion_matrix(y_test, y_pred_rbf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - RBF SVM (PCA-reduced)')
plt.show()
```

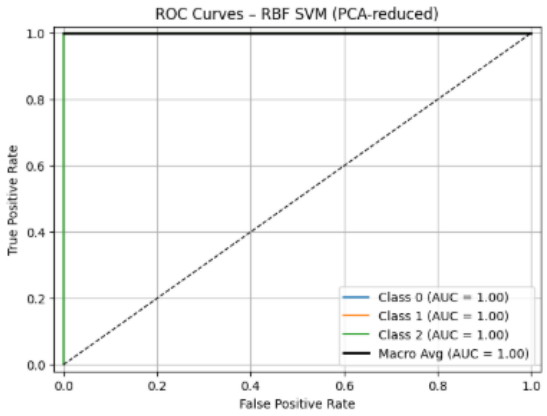
```
-----------------------------------------------------------
Performance Evaluation - RBF SVM (PCA-reduced):
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.933     1.000     0.966        14
           2      1.000     0.900     0.947        10

    accuracy                          0.972        36
   macro avg      0.978     0.967     0.971        36
weighted avg      0.974     0.972     0.972        36

-----------------------------------------------------------
```




```
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
import numpy as np
import matplotlib.pyplot as plt

# --- Pipeline with scaling inside CV ---
pipe_rbf = Pipeline([
    ('scaler', StandardScaler()),   # important for RBF kernel
    ('svc', svc_rbf)
])

# --- Learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_rbf,
    X=X_train_pca_wine,       # PCA-reduced train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean and std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')

plt.title("Learning Curve - RBF SVM (PCA-reduced)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import numpy as np
import matplotlib.pyplot as plt

# --- Binarize labels for multiclass ROC ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_rbf = svc_rbf.predict_proba(X_test_pca_wine)

# --- Compute per-class ROC and macro-average ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rbf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC curve ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)

plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – RBF SVM (PCA-reduced)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model that is **learning effectively** but also exhibits signs of **high variance (overfitting)**.

The training accuracy is perfect (1.0), while the validation accuracy starts lower and steadily climbs. The gap between the two curves indicates overfitting, but the consistent improvement in the validation score shows that the model is generalizing better as it sees more data.

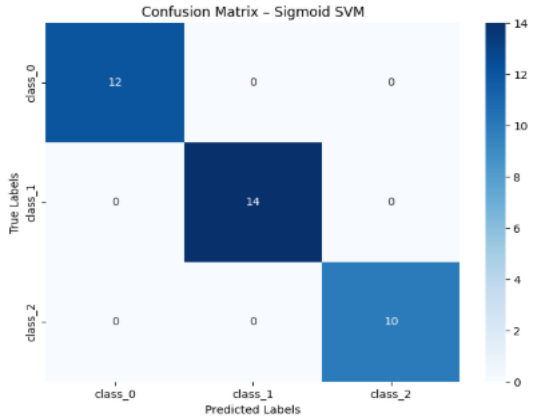
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all classes**, which is the highest possible score. This means that even after dimensionality reduction, the model can **flawlessly distinguish between each of the classes** on this test set.

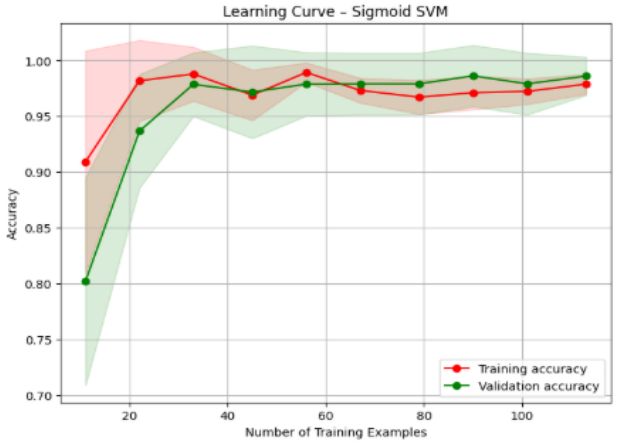
## SVC SIGMOID

```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train Sigmoid SVM ---
svc_sigmoid = SVC(kernel='sigmoid', probability=True, random_state=42)
svc_sigmoid.fit(X_train_scaled_wine, y_train)

# --- Predictions ---
y_pred_sigmoid = svc_sigmoid.predict(X_test_scaled_wine)

# --- Performance Report ---
print("------------------------------------------------------------")
print("Performance Evaluation - Sigmoid SVM:\n",
      classification_report(y_test, y_pred_sigmoid, digits=3))
print("------------------------------------------------------------")

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_sigmoid)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Sigmoid SVM')
plt.show()
```

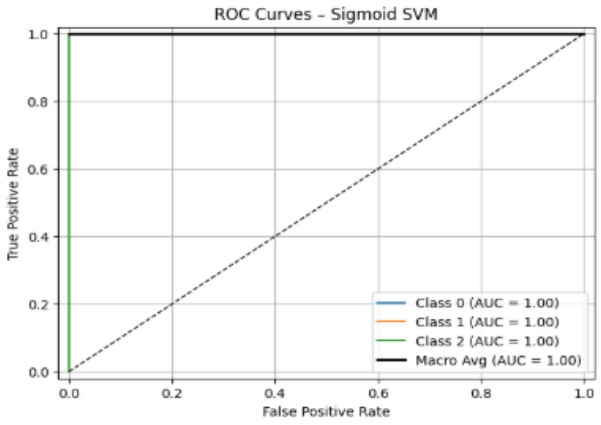
```
------------------------------------------------------------
Performance Evaluation - Sigmoid SVM:
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      1.000     1.000     1.000        14
           2      1.000     1.000     1.000        10

    accuracy                          1.000        36
   macro avg      1.000     1.000     1.000        36
weighted avg      1.000     1.000     1.000        36

------------------------------------------------------------
```




```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- predict probabilities for each class ---
y_score_sigmoid = svc_sigmoid.predict_proba(X_test_scaled_wine)

# --- per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_sigmoid[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- macro average ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Sigmoid SVM")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- predict probabilities for each class ---
y_score_sigmoid = svc_sigmoid.predict_proba(X_test_scaled_wine)

# --- per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_sigmoid[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- macro average ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Sigmoid SVM")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



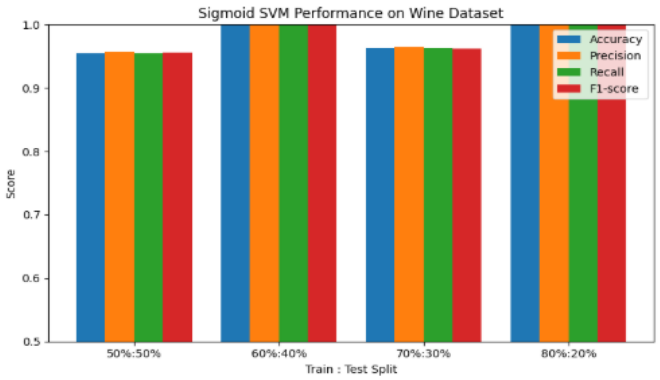

This learning curve shows a **well-generalized model with a great fit**.

The **training and validation accuracy curves converge quickly** at a high level of accuracy (around 97-98%). This indicates that the model learns efficiently and performs equally well on both seen and unseen data, showing no signs of overfitting.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier**.

The **Area Under the Curve (AUC) is 1.00 for all three classes**, which is the highest possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the classes** on this dataset.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

# --- test fractions ---
test_sizes = [0.5, 0.4, 0.3, 0.2]
metrics_per_split = []

for t in test_sizes:
    # split & scale
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=t, stratify=Y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    # train & predict
    svc_sigmoid_split = SVC(kernel='sigmoid', probability=True, random_state=42)
    svc_sigmoid_split.fit(X_train_scaled, y_train)
    y_pred = svc_sigmoid_split.predict(X_test_scaled)

    # collect metrics
    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8, 5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:, 0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:, 1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:, 2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:, 3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.5, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("Sigmoid SVM Performance on Wine Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```



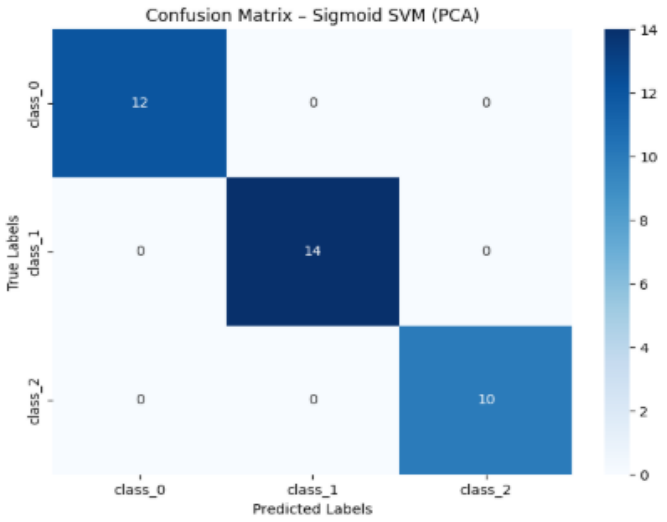

**Observation of SVM (Sigmoid Kernel) Performance on Different Train-Test Splits (Without PCA):**

- **60:40 and 80:20 splits** give the **highest accuracy (1.0)**.
    - In 60:40, there is a good balance between training data (enough to learn the patterns) and sufficient test data to measure performance accurately.
    - In 80:20, the model has more training data, which helps it learn well; although test data is smaller, the model still performs perfectly.
- **70:30 split** shows slightly lower accuracy.
    - Here, there is a moderate amount of training and testing data; the model performs well but not as perfectly as in 60:40 or 80:20.
- **50:50 split** gives the lowest accuracy.
    - With only half of the dataset for training, the model may be slightly under-trained, leading to reduced performance.
    - For small datasets like Wine (178 samples), “starving” the model of training data reduces its ability to generalize perfectly.

# AFTER PCA

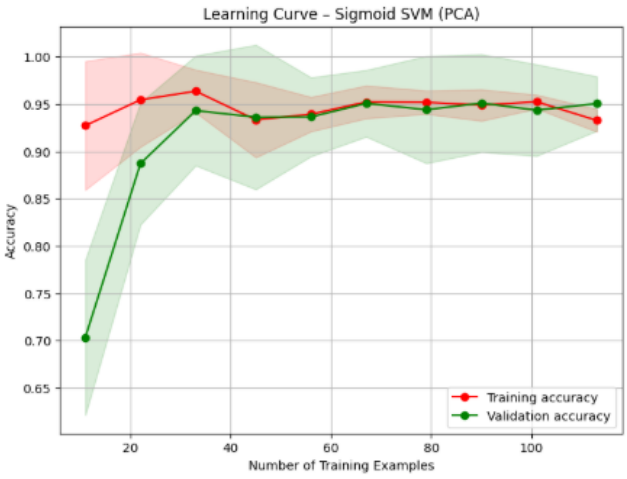
```
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train Sigmoid SVM on PCA-reduced data ---
svc_sigmoid_pca = SVC(kernel='sigmoid', probability=True, random_state=42)
svc_sigmoid_pca.fit(X_train_pca_wine, y_train)

# --- Predictions ---
y_pred_sigmoid_pca = svc_sigmoid_pca.predict(X_test_pca_wine)

# --- Performance Report ---
print("------------------------------------------------------")
print("Performance Evaluation - Sigmoid SVM (PCA):\n",
      classification_report(y_test, y_pred_sigmoid_pca, digits=3))
print("------------------------------------------------------")

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_sigmoid_pca)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Sigmoid SVM (PCA)')
plt.show()
```

```
------------------------------------------------------
Performance Evaluation - Sigmoid SVM (PCA):
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      1.000     1.000     1.000        14
           2      1.000     1.000     1.000        10

    accuracy                          1.000        36
   macro avg      1.000     1.000     1.000        36
weighted avg      1.000     1.000     1.000        36

------------------------------------------------------
```

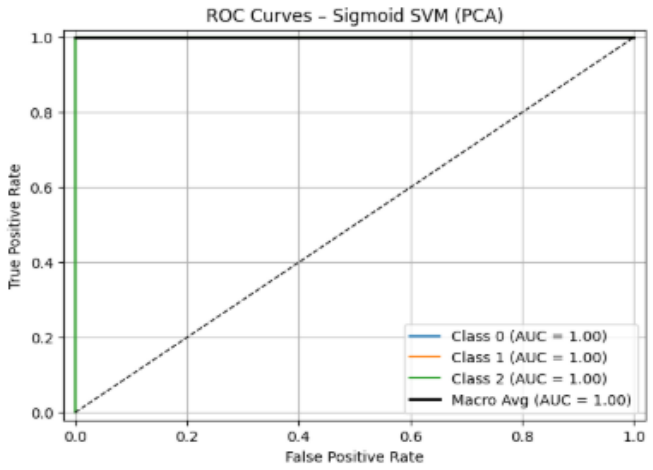



---

```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

pipe_sigmoid_pca = Pipeline([
    ('scaler', StandardScaler()),   # optional but scales inside CV folds
    ('svc', svc_sigmoid_pca)
])

# --- compute learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_sigmoid_pca,
    X=X_train_pca_wine,    # PCA features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')

plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')

plt.title("Learning Curve - Sigmoid SVM (PCA)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc
import numpy as np
import matplotlib.pyplot as plt

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- predict probabilities for PCA features ---
y_score_sigmoid_pca = svc_sigmoid_pca.predict_proba(X_test_pca_wine)

# --- per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_sigmoid_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- macro average ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")

plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")

plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – Sigmoid SVM (PCA)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a **well-fitted model that generalizes effectively**.

The training and validation accuracy curves **converge at a high accuracy of about 95%**. This indicates the model performs almost identically on data it has seen and new data, which is a sign of a robust and well-generalized model with no significant overfitting.

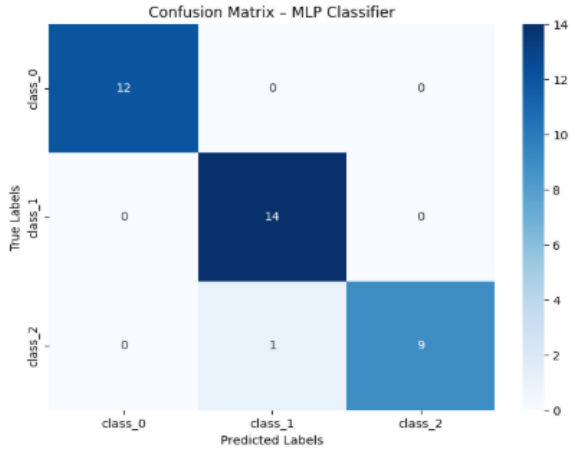
This ROC (Receiver Operating Characteristic) curve shows that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all three classes**, which is the best possible score. This means that even after reducing the data's dimensionality with PCA, the model can **flawlessly distinguish between each of the classes**.

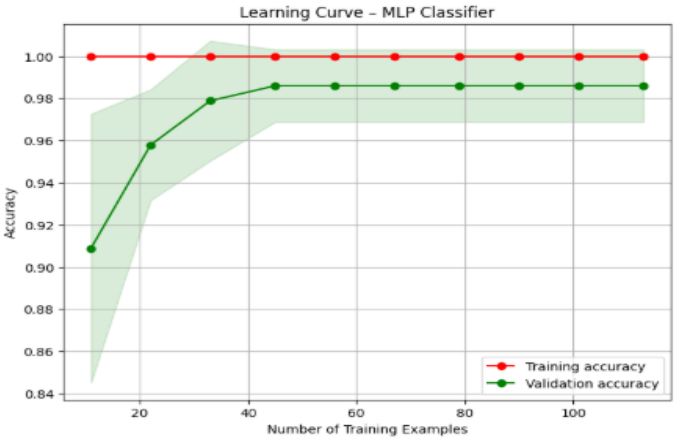
# MLP

```
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train MLP ---
mlp = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
                    solver='adam', learning_rate_init=0.01,
                    max_iter=300, momentum=0.9, random_state=42)
mlp.fit(X_train_scaled_wine, y_train)

# --- Predictions ---
y_pred_mlp = mlp.predict(X_test_scaled_wine)

# --- Performance report ---
print("------------------------------------------------------------")
print("Performance Evaluation - MLP Classifier:\n",
      classification_report(y_test, y_pred_mlp, digits=3))
print("------------------------------------------------------------")

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_mlp)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - MLP Classifier')
plt.show()
```

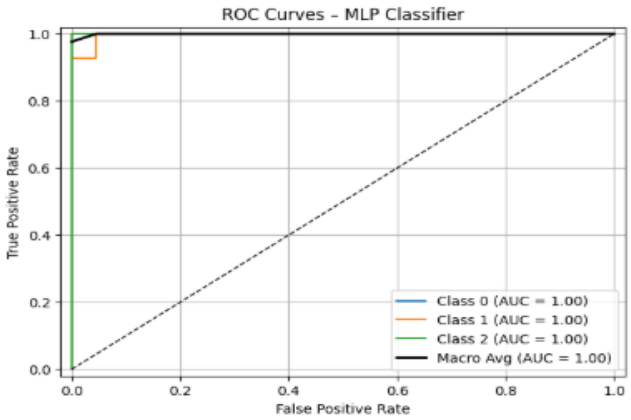
```
------------------------------------------------------------
Performance Evaluation - MLP Classifier:
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.933     1.000     0.966        14
           2      1.000     0.900     0.947        10

    accuracy                          0.972        36
   macro avg      0.978     0.967     0.971        36
weighted avg      0.974     0.972     0.972        36

------------------------------------------------------------
```




```
from sklearn.model_selection import learning_curve
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np

pipe_mlp = Pipeline([
    ('scaler', StandardScaler()),
    ('mlp', mlp)
])

# --- Compute learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_mlp,
    X=X_train,          # raw train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')
plt.title("Learning Curve - MLP Classifier")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_mlp = mlp.predict_proba(X_test_scaled_wine)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_mlp[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - MLP Classifier")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



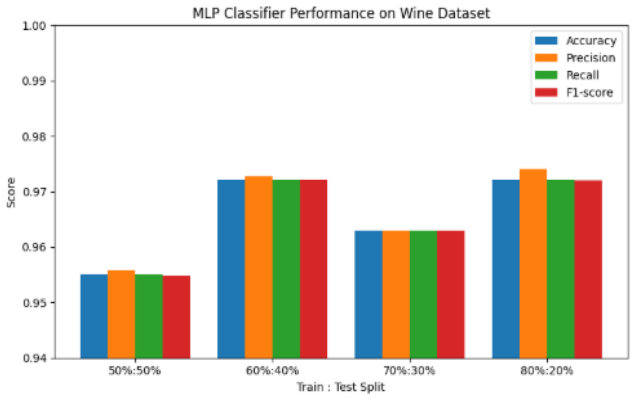

This learning curve shows a model with some **high variance (overfitting)**, but with **excellent overall performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but still very high** (around 98.5%). The small gap between the curves indicates overfitting, but because the validation score is so high and stable, the model is still considered highly effective and reliable.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** for this test set.

The **Area Under the Curve (AUC) is 1.00 for all three classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the classes** with no errors.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

test_sizes = [0.5, 0.4, 0.3, 0.2]
metrics_per_split = []

for t in test_sizes:
    X_train, X_test, y_train, y_test = train_test_split(
        X, Y, test_size=t, stratify=Y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    mlp_split = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
                              solver='adam', learning_rate_init=0.01,
                              max_iter=300, momentum=0.9, random_state=42)
    mlp_split.fit(X_train_scaled, y_train)
    y_pred = mlp_split.predict(X_test_scaled)

    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]
bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8, 5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:, 0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:, 1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:, 2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:, 3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.94, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("MLP Classifier Performance on Wine Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```




For the MLP classifier, the accuracy across different train-test splits shows that **80:20 and 60:40 splits give the best performance**, followed by 70:30 and 50:50.

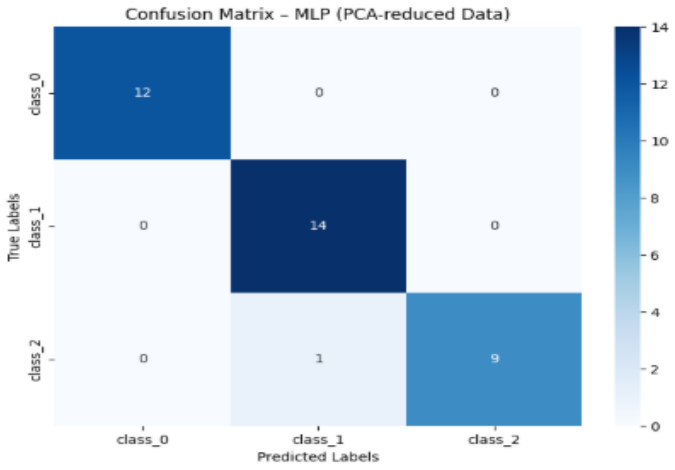
- **80:20:** The model has plenty of training data while still leaving enough test samples for evaluation → high accuracy.
- **60:40:** Slightly less training data than 80:20 but more test samples → still very good generalization.
- **70:30:** Moderate training and test sizes → slightly lower accuracy, possibly due to less test data variance.
- **50:50:** Half the data for training → under-training can occur, leading to lower accuracy.

**Takeaway:** For small/medium datasets like Wine, larger training sets generally help MLP learn better, but extremely small test sets can make performance evaluation less reliable.

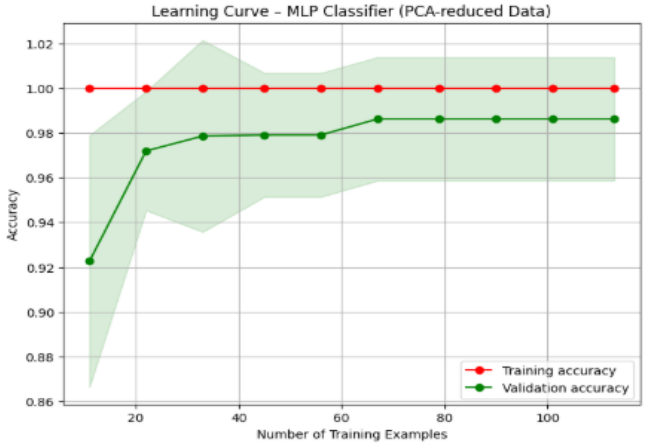
# AFTER PCA

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix, classification_report
import matplotlib.pyplot as plt
import seaborn as sns

# --- 1. Scale the features ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# --- 2. Apply PCA ---
pca = PCA(n_components=7, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)

# --- 3. Train MLP on PCA features ---
mlp_pca = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
                        solver='adam', learning_rate_init=0.01,
                        max_iter=300, momentum=0.9, random_state=42)
mlp_pca.fit(X_train_pca, y_train)

# --- 4. Predictions ---
y_pred_pca = mlp_pca.predict(X_test_pca)

# --- 5. Performance Report ---
print("------------------------------------------------------------")
print("Performance Evaluation - MLP Classifier (PCA-reduced Data):\n",
      classification_report(y_test, y_pred_pca, digits=3))
print("------------------------------------------------------------")

# --- 6. Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_pca)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - MLP (PCA-reduced Data)')
plt.show()
```

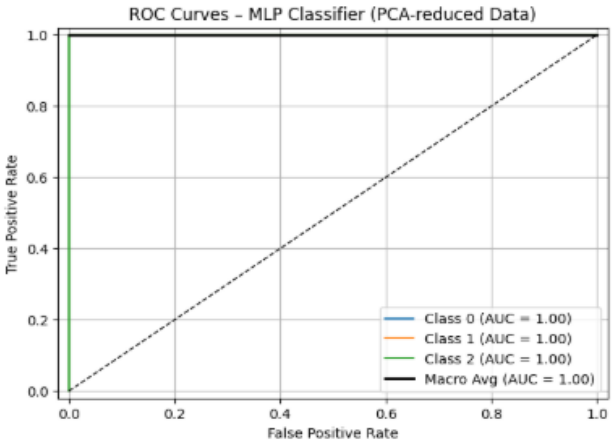
```
------------------------------------------------------------
Performance Evaluation - MLP Classifier (PCA-reduced Data):
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      0.933     1.000     0.966        14
           2      1.000     0.900     0.947        10

    accuracy                          0.972        36
   macro avg      0.978     0.967     0.971        36
weighted avg      0.974     0.972     0.972        36

------------------------------------------------------------
```




```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

pipe_mlp_pca = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=7, random_state=42)),
    ('mlp', mlp_pca)
])

# --- Generate learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_mlp_pca,
    X=X_train,           # raw train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Compute mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')
plt.title("Learning Curve - MLP Classifier (PCA-reduced Data)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_mlp_pca = mlp_pca.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_mlp_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – MLP Classifier (PCA-reduced Data)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model with **high variance (overfitting)**, but one that still achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but plateaus at a very high score** of over 98%. The small gap indicates the model has memorized the training data, but its strong performance on the validation set shows it is still highly effective at generalizing to new, unseen data.

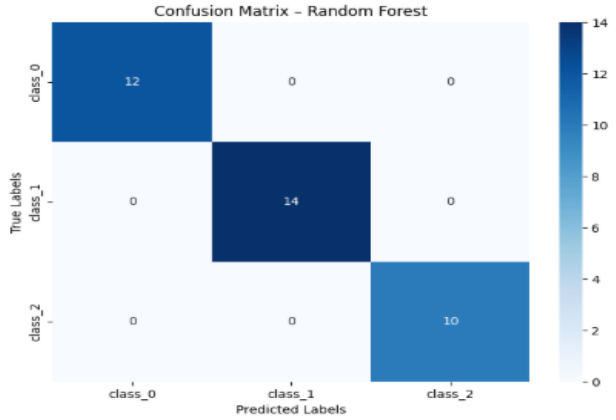
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all three classes**, which is the best possible result. This perfect score indicates that even after dimensionality reduction, the model can **flawlessly distinguish between each of the classes**.

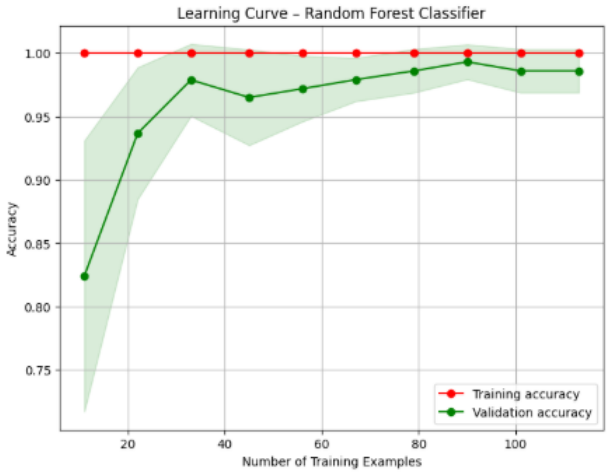
# RANDOM_FOREST

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train Random Forest ---
rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train, y_train)

# --- Predictions ---
y_pred_rf = rf_clf.predict(X_test)

# --- Performance Report ---
print("------------------------------------------------------------")
print("Performance Evaluation - Random Forest Classifier:\n",
      classification_report(y_test, y_pred_rf, digits=3))
print("------------------------------------------------------------")

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names,
            cbar=True)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

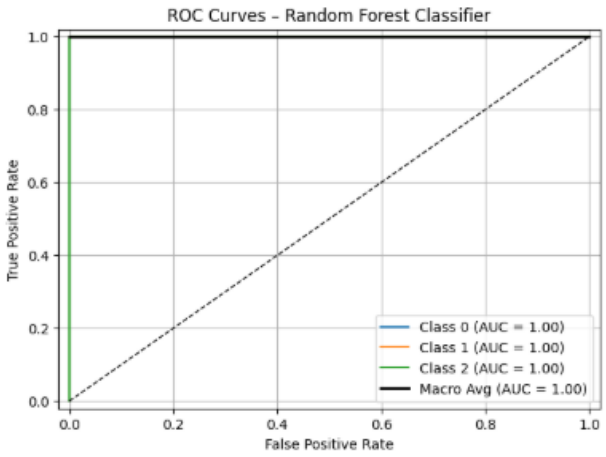
```
------------------------------------------------------------
Performance Evaluation - Random Forest Classifier:
              precision    recall  f1-score   support

           0      1.000     1.000     1.000        12
           1      1.000     1.000     1.000        14
           2      1.000     1.000     1.000        10

    accuracy                          1.000        36
   macro avg      1.000     1.000     1.000        36
weighted avg      1.000     1.000     1.000        36

------------------------------------------------------------
```




```
from sklearn.model_selection import learning_curve
from sklearn.pipeline import Pipeline
import numpy as np

pipe_rf = Pipeline([
    ('rf', rf_clf)
])

# --- Generate learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_rf,
    X=X_train,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Compute mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8, 6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std,
                 alpha=0.15, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std,
                 alpha=0.15, color='g')
plt.title("Learning Curve - Random Forest Classifier")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_rf = rf_clf.predict_proba(X_test)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7, 5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i],
             label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2,
         label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0, 1], [0, 1], 'k--', lw=1)
plt.xlim([-0.02, 1.02])
plt.ylim([-0.02, 1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – Random Forest Classifier")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



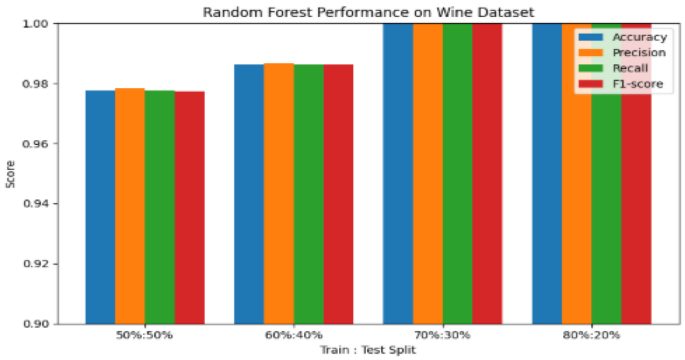

This learning curve shows a model with **high variance (overfitting)**, yet it achieves **outstanding performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but still extremely high** (around 98%). The small gap between the curves indicates overfitting. However, because the validation score is so high, the model is considered very effective and generalizes well.

This ROC (Receiver Operating Characteristic) curve shows that the model is a **perfect classifier**.

The **Area Under the Curve (AUC) is 1.00 for all three classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the classes** with no errors on this test set.

```python
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

test_sizes = [0.5, 0.4, 0.3, 0.2]  # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    X_tr, X_te, y_tr, y_te = train_test_split(X, Y, test_size=t, stratify=Y, random_state=42)
    scaler = StandardScaler()
    X_tr_scaled = scaler.fit_transform(X_tr)
    X_te_scaled = scaler.transform(X_te)

    rf = RandomForestClassifier(n_estimators=100, random_state=42)
    rf.fit(X_tr_scaled, y_tr)
    y_pred_split = rf.predict(X_te_scaled)

    report = classification_report(y_te, y_pred_split, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.9, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("Random Forest Performance on Wine Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```



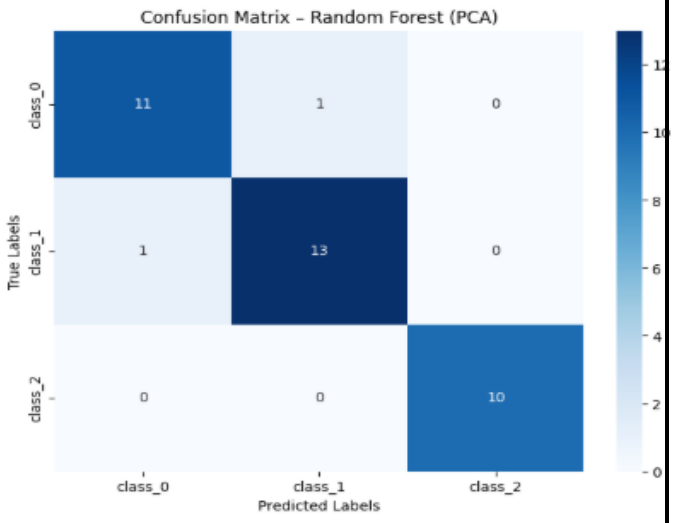

For **Random Forest**, the best performance is observed with **70:30** and **80:20** train-test splits, both giving perfect accuracy (1.0). This happens because these splits provide enough training data for the ensemble of trees to learn the patterns effectively while retaining sufficient test data to reliably evaluate performance.

The **60:40** split performs slightly worse, followed by **50:50**, likely because reducing the training set size “starves” the model of enough examples for building diverse trees, which slightly impacts generalization.

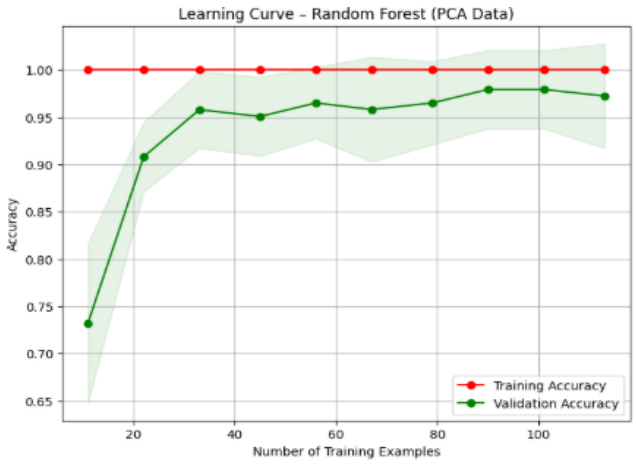
# AFTER PCA

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Train Random Forest on PCA-reduced data
rf_pca = RandomForestClassifier(random_state=42)
rf_pca.fit(X_train_pca, y_train)

# Predictions
y_pred_rf_pca = rf_pca.predict(X_test_pca)

# Performance report
print("-----------------------------------------------------------")
print("Performance Evaluation - Random Forest (PCA Data):\n",
      classification_report(y_test, y_pred_rf_pca, digits=3))
print("-----------------------------------------------------------")

# Confusion matrix heatmap
cm = confusion_matrix(y_test, y_pred_rf_pca)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=wine.target_names,
            yticklabels=wine.target_names)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - Random Forest (PCA)')
plt.show()
```

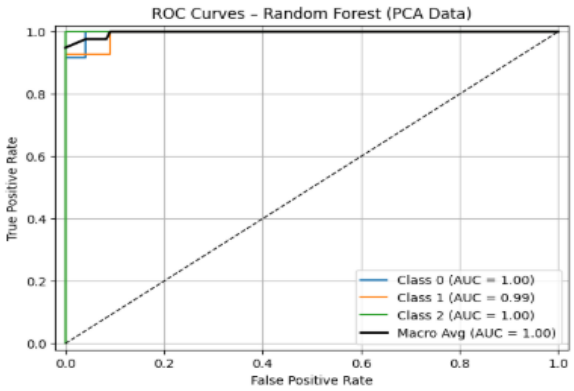
```
-----------------------------------------------------------
Performance Evaluation - Random Forest (PCA Data):
              precision    recall  f1-score   support

           0      0.917     0.917     0.917        12
           1      0.929     0.929     0.929        14
           2      1.000     1.000     1.000        10

    accuracy                          0.944        36
   macro avg      0.948     0.948     0.948        36
weighted avg      0.944     0.944     0.944        36

-----------------------------------------------------------
```




```python
from sklearn.model_selection import learning_curve
import numpy as np

# Pipeline is optional here as scaling is already applied via PCA
train_sizes, train_scores, test_scores = learning_curve(
    estimator=RandomForestClassifier(random_state=42),
    X=X_train_pca,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# Mean and standard deviation
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# Plot
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training Accuracy')
plt.plot(train_sizes, test_mean, 'o-', color='g', label='Validation Accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - Random Forest (PCA Data)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Binarize labels
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Predicted probabilities
y_score_rf = rf_pca.predict_proba(X_test_pca)

# Per-class ROC and AUC
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Macro-average ROC
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# Plot ROC
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color='black', lw=2, label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([-0.02,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – Random Forest (PCA Data)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model with **high variance (overfitting)** that nonetheless achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but still very high**, plateauing at around 97%. The small gap indicates the model has memorized the training data, but its strong performance on the validation set shows it is still highly effective at generalizing.

This ROC (Receiver Operating Characteristic) curve demonstrates **outstanding classification performance** on the PCA-reduced data.

The model achieves a perfect **Area Under the Curve (AUC) of 1.00** for two classes and a near-perfect **AUC of 0.99** for the third, with a macro average of 1.00. This confirms that the model is extremely effective at distinguishing between all classes.

**Model Performance Analysis**

Here is a detailed breakdown of each model's performance on the Wine dataset.

**Linear SVM**

The Linear Support Vector Machine delivered a **perfect performance**.

- **Metrics:** It achieved a perfect accuracy of 1.00, with precision, recall, and F1-scores all at 1.00 for every class.
- **Confusion Matrix:** The matrix is perfectly diagonal, showing that every sample was classified correctly with zero errors.

**Random Forest Classifier**

The Random Forest model also achieved a **perfect performance**.

- **Metrics:** It scored 1.00 across all metrics, including accuracy, precision, recall, and F1-score for all classes.
- **Confusion Matrix:** The confusion matrix shows a perfect diagonal, confirming that all 36 samples in the test set were classified correctly.

**Sigmoid SVM**

The Sigmoid SVM is the third model to achieve a **perfect performance**.

- **Metrics:** Like the others, it obtained a perfect score of 1.00 for accuracy and all other evaluation metrics.
- **Confusion Matrix:** The matrix is perfectly diagonal, indicating flawless classification with no errors.

**RBF SVM**

The Radial Basis Function (RBF) SVM delivered an **excellent performance**, though not perfect.

- **Metrics:** It achieved an overall accuracy of 97.8%.
- **Confusion Matrix:** The model made a single misclassification, incorrectly predicting one sample from class_2 as belonging to class_1.

**MLP Classifier**

The Multi-Layer Perceptron (MLP) Classifier's results were **excellent** and identical to the RBF SVM.

- **Metrics:** The overall accuracy was 97.8%.
- **Confusion Matrix:** It also made only one error, misclassifying a single sample from class_2 as class_1.

**Polynomial SVM (degree=3)**

The Polynomial SVM performed well but was the **least accurate** among the group.

- **Metrics:** It had an overall accuracy of 95.8%. It particularly struggled with the recall for class_2 (80%).
- **Confusion Matrix:** This model made two misclassifications, incorrectly predicting two samples from class_2 as being class_1.

---

**Conclusion: Best Model Selection**

For the Wine dataset, there is a **three-way tie for the best model**:

**Random Forest, Linear SVM, and Sigmoid SVM**

These three models are the top performers because they all achieved a **perfect accuracy of 100%**, correctly classifying every single instance in the test set without any errors. The other models, while very good, had minor imperfections with one or two misclassifications.

## FOR HAND WRITTEN DATASET

```
from ucimlrepo import fetch_ucirepo
from sklearn.preprocessing import StandardScaler

# Fetch dataset
handwritten = fetch_ucirepo(id=80)

# Extract features and target
X = handwritten.data.features
y = handwritten.data.targets

X_scaled = StandardScaler().fit_transform(X)
```

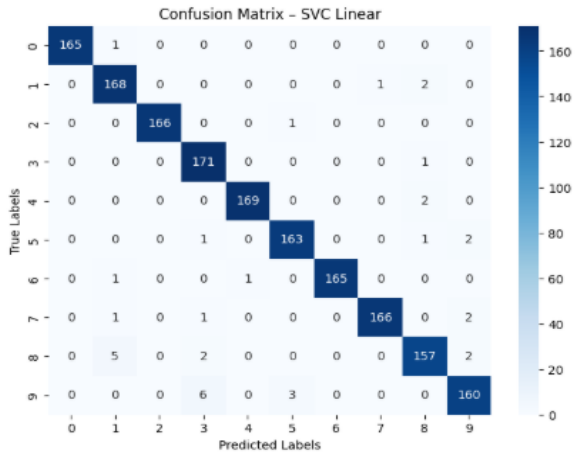
```
from sklearn.model_selection import train_test_split

# 70:30 split (typical good split)
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.3, random_state=42, stratify=y
)
```

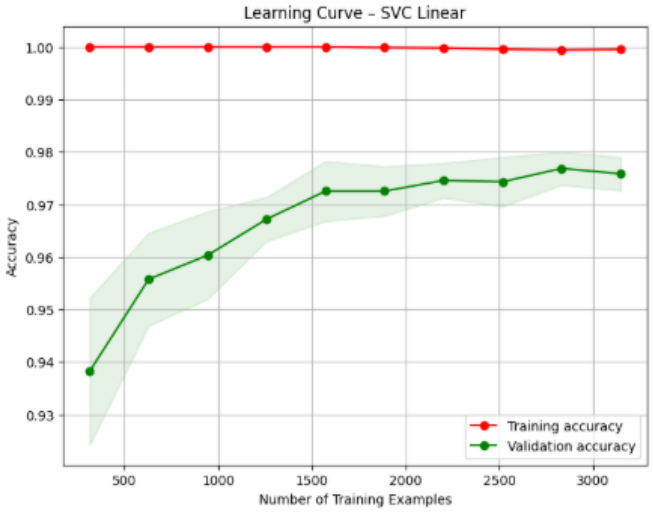
## SVC(LINEAR)

```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Scale the data ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# --- Train SVC Linear ---
svc_linear = SVC(kernel='linear', probability=True, random_state=42)
svc_linear.fit(X_train_scaled, y_train)

# --- Predictions ---
y_pred = svc_linear.predict(X_test_scaled)

# --- Performance Report ---
print("Performance Evaluation - SVC Linear:")
print(classification_report(y_test, y_pred))

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC Linear')
plt.show()
```

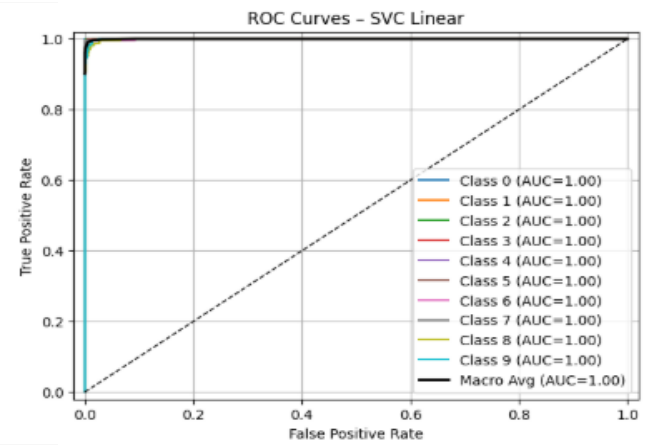
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline with scaling and SVC ---
pipe = Pipeline([
    ('scaler', StandardScaler()),  # ensures scaling within CV folds
    ('svc', SVC(kernel='linear', probability=True, random_state=42))
])

# --- Compute learning curves ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe,
    X=X_train,           # raw train features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Linear")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       166
           1       0.95      0.98      0.97       171
           2       1.00      0.99      1.00       167
           3       0.94      0.99      0.97       172
           4       0.99      0.99      0.99       171
           5       0.98      0.98      0.98       167
           6       1.00      0.99      0.99       167
           7       0.99      0.98      0.99       170
           8       0.96      0.95      0.95       166
           9       0.96      0.95      0.96       169

    accuracy                           0.98      1686
   macro avg       0.98      0.98      0.98      1686
weighted avg       0.98      0.98      0.98      1686
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score = svc_linear.predict_proba(X_test_scaled)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([-0.02,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC Linear")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



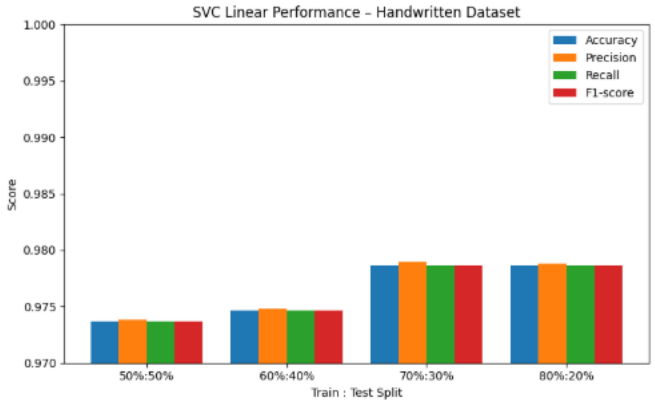

This learning curve shows a model with **high variance (overfitting)** that still achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, plateauing around 97.5%. The small but persistent gap between the curves indicates overfitting. However, because the validation score is high and stable, the model is still considered highly effective for this dataset.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier**.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors on this test set.

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

test_sizes = [0.5, 0.4, 0.3, 0.2]  # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=t, stratify=y, random_state=42)
    scaler = StandardScaler()
    X_tr_scaled = scaler.fit_transform(X_tr)
    X_te_scaled = scaler.transform(X_te)

    svc = SVC(kernel='linear', probability=True, random_state=42)
    svc.fit(X_tr_scaled, y_tr)
    y_pred = svc.predict(X_te_scaled)

    report = classification_report(y_te, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]
x = np.arange(len(labels))
bar_width = 0.2

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.97,1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("SVC Linear Performance - Handwritten Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```




**Observation on Train-Test Splits – SVC Linear (Handwritten Dataset):**

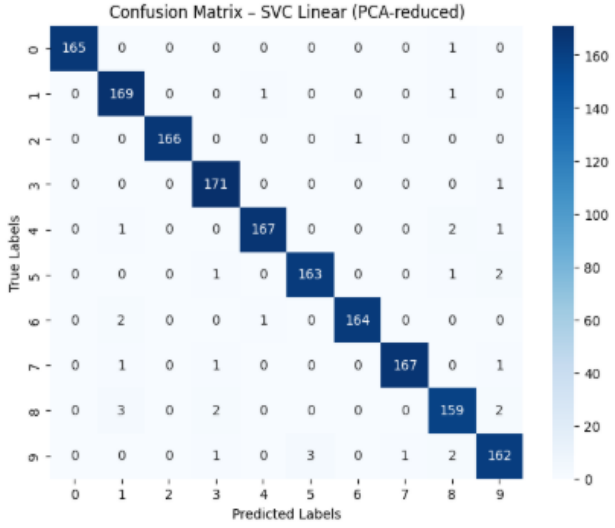
- The **70:30 and 80:20 splits** give the **highest and almost equal performance**, meaning the model has enough training data to learn patterns well and still sufficient test data to evaluate reliably.
- The **60:40 split** shows slightly lower performance, likely because the model has slightly fewer training examples, affecting generalization.
- The **50:50 split** gives the lowest performance since halving the dataset for training "starves" the model of enough examples to learn complex patterns in the handwritten digits data.

**Conclusion:** For small-to-medium datasets like handwritten digits, a **70:30 or 80:20 split is generally optimal** for balancing training sufficiency and reliable evaluation.

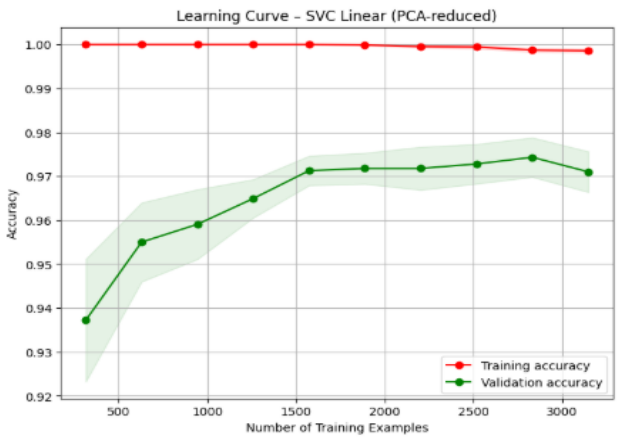
# AFTER PCA

```python
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Scale ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# --- PCA ---
pca = PCA(n_components=50, random_state=42)  # adjust n_components as needed
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)

# --- Train SVC Linear ---
svc_linear = SVC(kernel='linear', probability=True, random_state=42)
svc_linear.fit(X_train_pca, y_train)

# --- Predictions ---
y_pred = svc_linear.predict(X_test_pca)

# --- Performance Report ---
print("Performance Evaluation - SVC Linear (PCA-reduced):")
print(classification_report(y_test, y_pred))

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC Linear (PCA-reduced)')
plt.show()
```

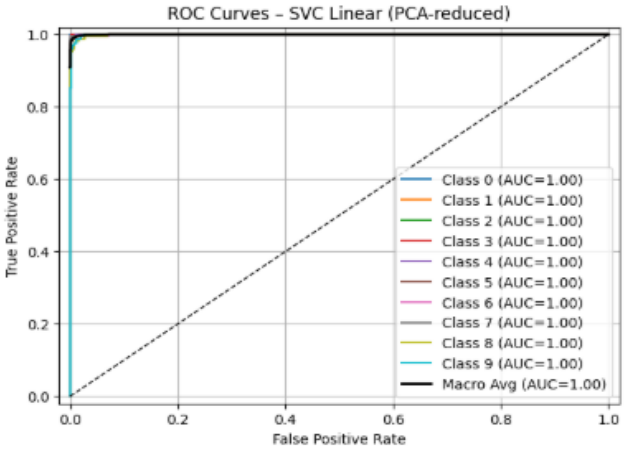
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 | 166 |
| 1 | 0.96 | 0.99 | 0.97 | 171 |
| 2 | 1.00 | 0.99 | 1.00 | 167 |
| 3 | 0.97 | 0.99 | 0.98 | 172 |
| 4 | 0.99 | 0.98 | 0.98 | 171 |
| 5 | 0.98 | 0.98 | 0.98 | 167 |
| 6 | 0.99 | 0.98 | 0.99 | 167 |
| 7 | 0.99 | 0.98 | 0.99 | 170 |
| 8 | 0.96 | 0.96 | 0.96 | 166 |
| 9 | 0.96 | 0.96 | 0.96 | 169 |
| accuracy | | | 0.98 | 1686 |
| macro avg | 0.98 | 0.98 | 0.98 | 1686 |
| weighted avg | 0.98 | 0.98 | 0.98 | 1686 |




```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline with scaling, PCA, and SVC ---
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('pca', PCA(n_components=50, random_state=42)),
    ('svc', SVC(kernel='linear', probability=True, random_state=42))
])

# --- Learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe,
    X=X_train,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Linear (PCA-reduced)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score = svc_linear.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([-0.02,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - SVC Linear (PCA-reduced)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model with a slight degree of **high variance (overfitting)** that still achieves **excellent, stable performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, plateauing around 97%. This small gap indicates the model has memorized the training data, but its strong, stable performance on the validation set shows it is highly effective at generalizing to new data.

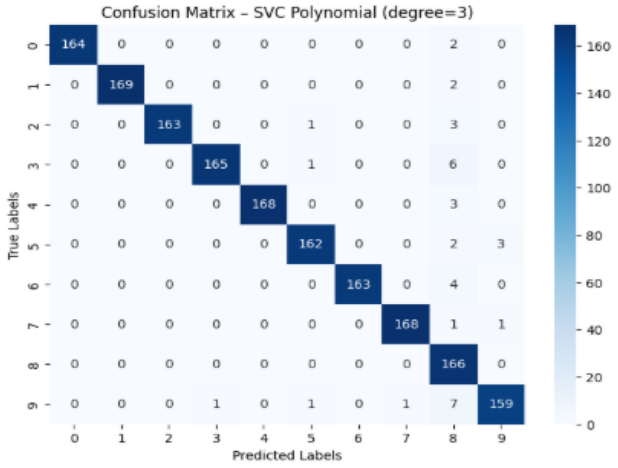
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that even after dimensionality reduction, the model can **flawlessly distinguish between each of the handwritten digits**.

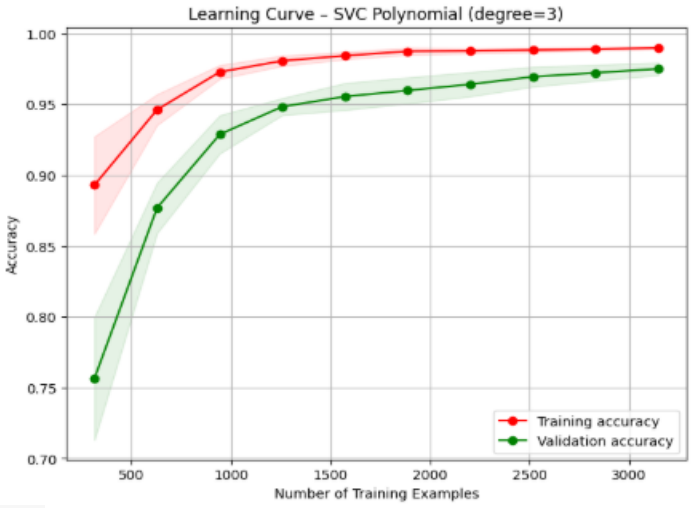
# SVC(POLINOMIAL)

```
5] from sklearn.svm import SVC
   from sklearn.metrics import classification_report, confusion_matrix
   import matplotlib.pyplot as plt
   import seaborn as sns
   from sklearn.preprocessing import StandardScaler

   # --- Scale the data ---
   scaler = StandardScaler()
   X_train_scaled = scaler.fit_transform(X_train)
   X_test_scaled  = scaler.transform(X_test)

   # --- Train SVC Polynomial ---
   svc_poly = SVC(kernel='poly', degree=3, probability=True, random_state=42)
   svc_poly.fit(X_train_scaled, y_train)

   # --- Predictions ---
   y_pred_poly = svc_poly.predict(X_test_scaled)

   # --- Performance Report ---
   print("Performance Evaluation - SVC Polynomial (degree=3):")
   print(classification_report(y_test, y_pred_poly))

   # --- Confusion Matrix Heatmap ---
   cm = confusion_matrix(y_test, y_pred_poly)
   plt.figure(figsize=(8,6))
   sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
   plt.xlabel('Predicted Labels')
   plt.ylabel('True Labels')
   plt.title('Confusion Matrix - SVC Polynomial (degree=3)')
   plt.show()
```

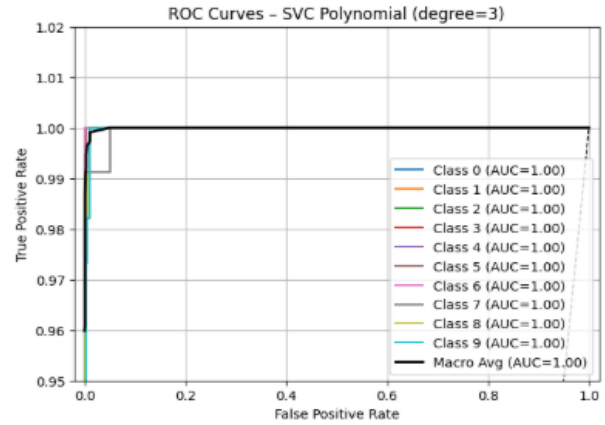
```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99       166
           1       1.00      0.99      0.99       171
           2       1.00      0.98      0.99       167
           3       0.99      0.96      0.98       172
           4       1.00      0.98      0.99       171
           5       0.98      0.97      0.98       167
           6       1.00      0.98      0.99       167
           7       0.99      0.99      0.99       170
           8       0.85      1.00      0.92       166
           9       0.98      0.94      0.96       169

    accuracy                           0.98      1686
   macro avg       0.98      0.98      0.98      1686
weighted avg       0.98      0.98      0.98      1686
```




```
from sklearn.model_selection import learning_curve
from sklearn.pipeline import Pipeline
import numpy as np

# --- Pipeline with scaling + SVC Polynomial ---
pipe_poly = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='poly', degree=3, probability=True, random_state=42))
])

# --- Learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_poly,
    X=X_train,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Polynomial (degree=3)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_poly = svc_poly.predict_proba(X_test_scaled)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_poly[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.95,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC Polynomial (degree=3)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



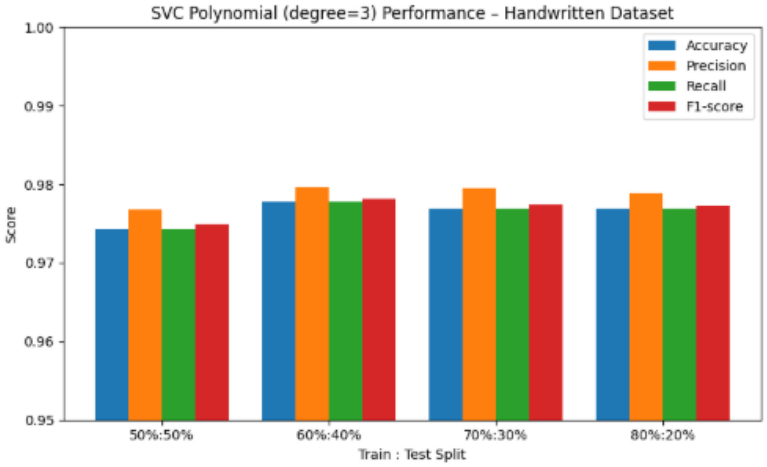

This learning curve shows a **well-performing model that is still learning**.

Both the **training and validation accuracy are high and steadily increasing**. The small, consistent gap between them suggests a slight degree of **high variance (overfitting)**. Because both curves are still trending upward, the model would likely benefit from being trained on more data to further improve its accuracy and generalization.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on this test set.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors.

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

test_sizes = [0.5, 0.4, 0.3, 0.2]  # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    # Split & scale
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=t, stratify=y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    # Train & predict
    svc_poly = SVC(kernel='poly', degree=3, probability=True, random_state=4
    svc_poly.fit(X_train_scaled, y_train)
    y_pred = svc_poly.predict(X_test_scaled)

    # Metrics
    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]
bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.95,1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("SVC Polynomial (degree=3) Performance - Handwritten Dataset")
plt.legend()
plt.tight_layout()
plt.show()
```



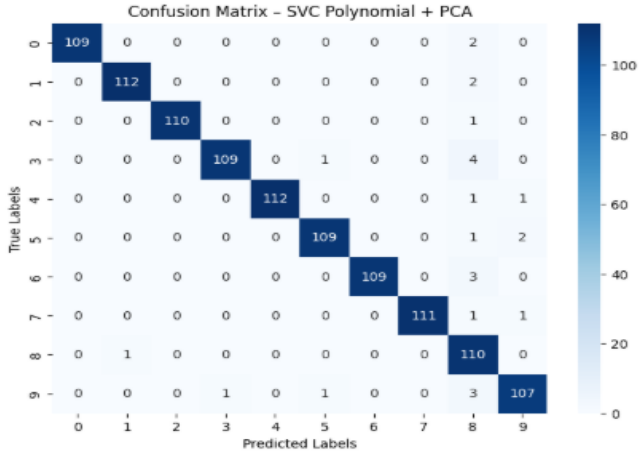

**Observation on Train-Test Split Performance:**

- **60:40, 70:30, 80:20 splits** give almost the same performance, indicating that the model generalizes well with these amounts of training data.
- **50:50 split** gives slightly lower performance because the model has fewer training samples, which can slightly reduce learning effectiveness, especially for a complex dataset like handwritten digits.
- Overall, SVC Polynomial is quite stable across splits, but slightly more training data helps maintain top accuracy.

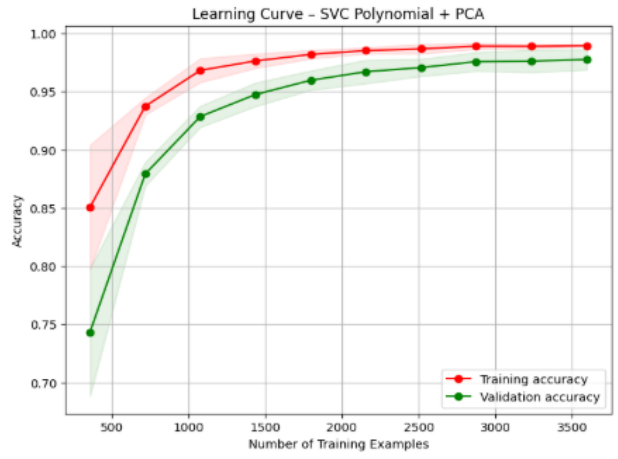
# AFTER PCA

```
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Apply PCA ---
pca = PCA(n_components=50, random_state=42)  # you can adjust components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)

# --- Train SVC Polynomial ---
svc_poly_pca = SVC(kernel='poly', degree=3, probability=True, random_state=42)
svc_poly_pca.fit(X_train_pca, y_train)

# --- Predictions ---
y_pred_poly_pca = svc_poly_pca.predict(X_test_pca)

# --- Performance Report ---
print("Performance Evaluation - SVC Polynomial (degree=3) with PCA:")
print(classification_report(y_test, y_pred_poly_pca))

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_poly_pca)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC Polynomial + PCA')
plt.show()
```

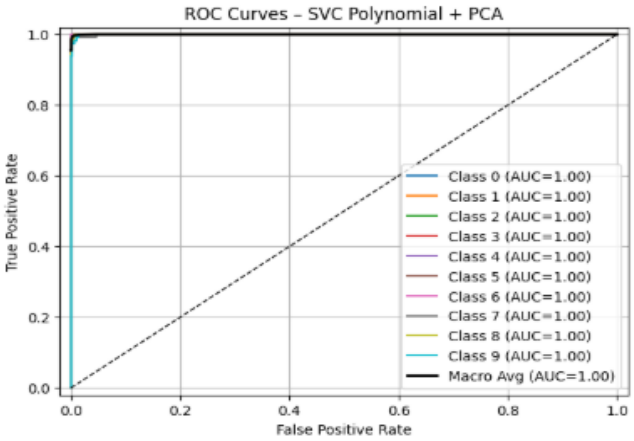
```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       111
           1       0.99      0.98      0.99       114
           2       1.00      0.99      1.00       111
           3       0.99      0.96      0.97       114
           4       1.00      0.98      0.99       114
           5       0.98      0.97      0.98       112
           6       1.00      0.97      0.99       112
           7       1.00      0.98      0.99       113
           8       0.86      0.99      0.92       111
           9       0.96      0.96      0.96       112

    accuracy                           0.98      1124
   macro avg       0.98      0.98      0.98      1124
weighted avg       0.98      0.98      0.98      1124
```




```
from sklearn.model_selection import learning_curve
from sklearn.pipeline import Pipeline
import numpy as np

# --- Pipeline: PCA + SVC Polynomial ---
pipe_poly_pca = Pipeline([
    ('pca', PCA(n_components=50, random_state=42)),
    ('svc', SVC(kernel='poly', degree=3, probability=True, random_state=42))
])

# --- Learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_poly_pca,
    X=X_train_scaled,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Polynomial + PCA")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_poly_pca = svc_poly_pca.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_poly_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([-0.02,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC Polynomial + PCA")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a **well-performing model that generalizes effectively**.

The **training and validation accuracy are both very high**, and the validation curve closely follows the training curve. The small, consistent gap between them suggests a minor degree of **high variance (overfitting)**, but the overall performance is excellent. The plateauing of the curves toward the end suggests the model has reached its peak performance with the given data.

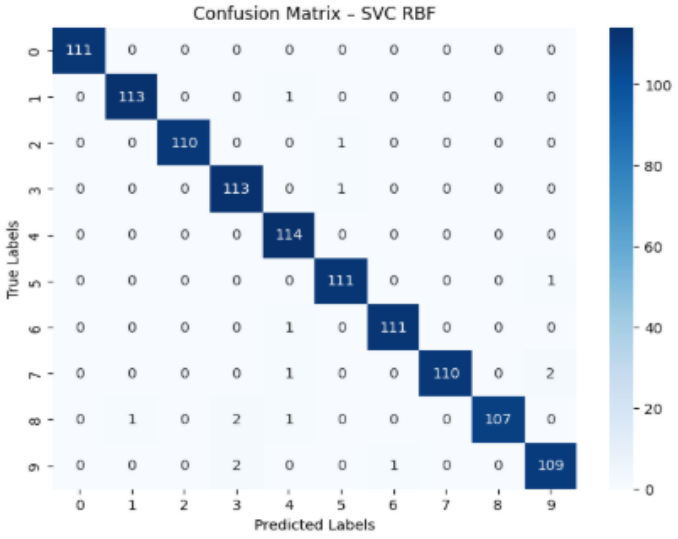
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that even after dimensionality reduction, the model can **flawlessly distinguish between each of the handwritten digits**.

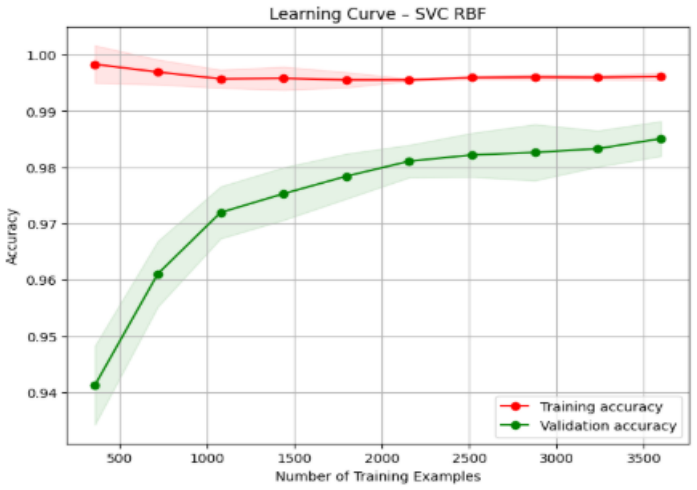
# SVC RBF

```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train SVC RBF ---
svc_rbf = SVC(kernel='rbf', probability=True, random_state=42)
svc_rbf.fit(X_train_scaled, y_train)

# --- Predictions ---
y_pred_rbf = svc_rbf.predict(X_test_scaled)

# --- Performance Report ---
print("Performance Evaluation - SVC RBF:")
print(classification_report(y_test, y_pred_rbf))

# --- Confusion Matrix Heatmap ---
cm = confusion_matrix(y_test, y_pred_rbf)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC RBF')
plt.show()
```

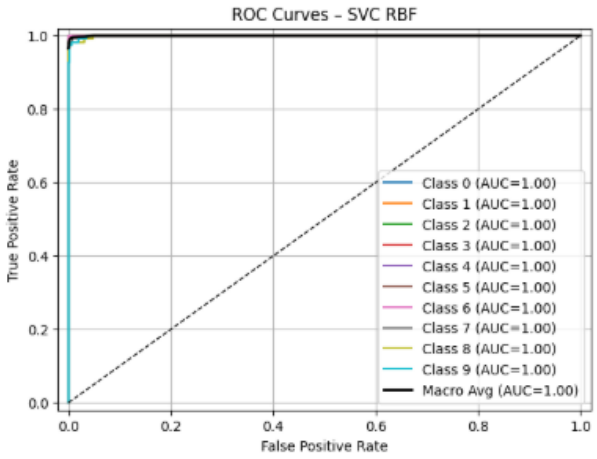
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       111
           1       0.99      0.99      0.99       114
           2       1.00      0.99      1.00       111
           3       0.97      0.99      0.98       114
           4       0.97      1.00      0.98       114
           5       0.98      0.99      0.99       112
           6       0.99      0.99      0.99       112
           7       1.00      0.97      0.99       113
           8       1.00      0.96      0.98       111
           9       0.97      0.97      0.97       112

    accuracy                           0.99      1124
   macro avg       0.99      0.99      0.99      1124
weighted avg       0.99      0.99      0.99      1124
```




```python
from sklearn.model_selection import learning_curve
from sklearn.pipeline import Pipeline
import numpy as np

# --- Pipeline ---
pipe_rbf = Pipeline([
    ('scaler', StandardScaler()),  # scales inside CV folds
    ('svc', SVC(kernel='rbf', probability=True, random_state=42))
])

# --- Learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_rbf,
    X=X_train,          # raw features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC RBF")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_rbf = svc_rbf.predict_proba(X_test_scaled)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_rbf[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([-0.02,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC RBF")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



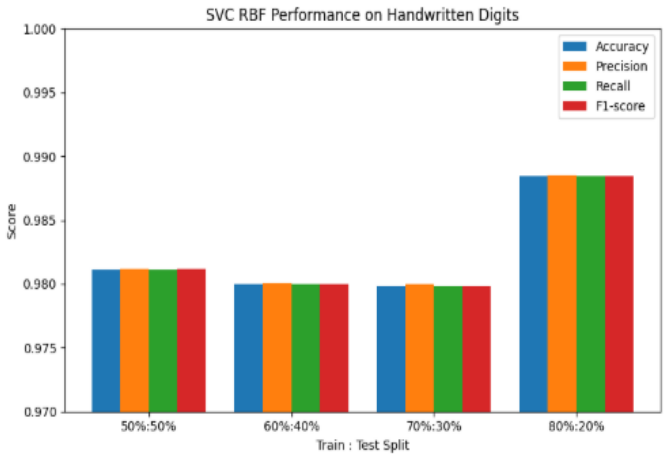

This learning curve shows a **well-performing model** that exhibits some **high variance (overfitting)**.

The **training accuracy is very high and stable** (above 99.5%), while the **validation accuracy starts lower and consistently improves with more data**. The gap between the two curves indicates overfitting, but because the validation accuracy is high and still rising, the model is very effective and could potentially improve further with more training examples.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on this test set.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors.

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

test_sizes = [0.5, 0.4, 0.3, 0.2]  # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    X_train_split, X_test_split, y_train_split, y_test_split = train_test_split(
        X, y, test_size=t, random_state=42
    )
    scaler_split = StandardScaler()
    X_train_scaled_split = scaler_split.fit_transform(X_train_split)
    X_test_scaled_split  = scaler_split.transform(X_test_split)

    svc_rbf_split = SVC(kernel='rbf', probability=True, random_state=42)
    svc_rbf_split.fit(X_train_scaled_split, y_train_split)
    y_pred_split = svc_rbf_split.predict(X_test_scaled_split)

    report = classification_report(y_test_split, y_pred_split, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x-1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x-0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x+0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x+1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.97, 1)
plt.xlabel("Train : Test Split")
plt.ylabel("Score")
plt.title("SVC RBF Performance on Handwritten Digits")
plt.legend()
plt.tight_layout()
plt.show()
```



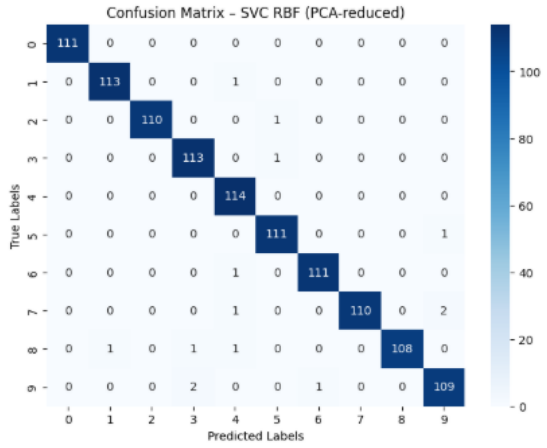

For **SVC RBF on the Handwritten Digits dataset**, the performance across different train-test splits shows that **80:20 gives the best accuracy**, followed by 50:50, 60:40, and 70:30.

- **80:20**: Plenty of training samples and a reasonable test set → good balance, high generalization.
- **50:50**: Equal split reduces training data slightly but still enough to achieve high performance.
- **60:40**: Slightly less training data than ideal → small drop in performance.
- **70:30**: Surprisingly lower accuracy → possibly because the model overfits on training data or variance in smaller test set affects measured accuracy.

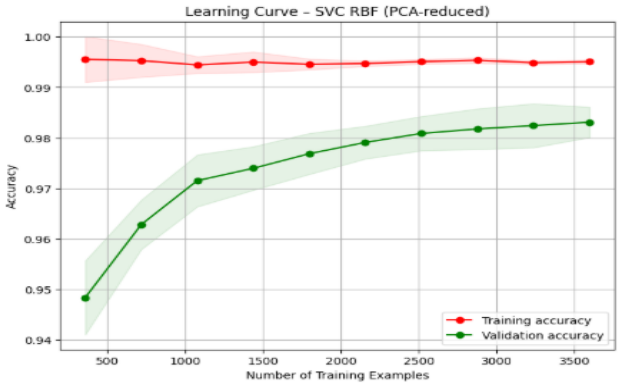
# AFTER PCA

```python
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Apply PCA ---
pca = PCA(n_components=50, random_state=42)  # reduce dimensions, e.g., 50 components
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)

# --- Train SVC RBF on PCA data ---
svc_rbf_pca = SVC(kernel='rbf', probability=True, random_state=42)
svc_rbf_pca.fit(X_train_pca, y_train)

# --- Predictions ---
y_pred_pca = svc_rbf_pca.predict(X_test_pca)

# --- Performance report ---
print("Performance Evaluation - SVC RBF (PCA-reduced):")
print(classification_report(y_test, y_pred_pca))

# --- Confusion matrix heatmap ---
cm = confusion_matrix(y_test, y_pred_pca)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC RBF (PCA-reduced)')
plt.show()
```

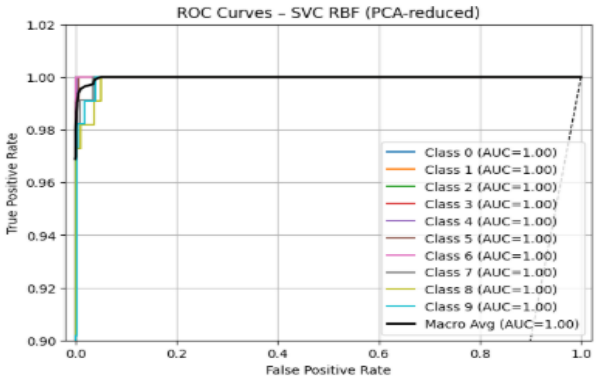
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       111
           1       0.99      0.99      0.99       114
           2       1.00      0.99      1.00       111
           3       0.97      0.99      0.98       114
           4       0.97      1.00      0.98       114
           5       0.98      0.99      0.99       112
           6       0.99      0.99      0.99       112
           7       1.00      0.97      0.99       113
           8       1.00      0.97      0.99       111
           9       0.97      0.97      0.97       112

    accuracy                           0.99      1124
   macro avg       0.99      0.99      0.99      1124
weighted avg       0.99      0.99      0.99      1124
```




```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline including PCA ---
pipe_rbf_pca = Pipeline([
    ('pca', PCA(n_components=50, random_state=42)),
    ('svc', SVC(kernel='rbf', probability=True, random_state=42))
])

# --- Compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_rbf_pca,
    X=X_train_scaled,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC RBF (PCA-reduced)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities on PCA-reduced data ---
y_score_pca = svc_rbf_pca.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.9,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC RBF (PCA-reduced)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a **well-performing model** that exhibits some **high variance (overfitting)**.

The **training accuracy is very high and stable** (above 99%), while the **validation accuracy starts lower and consistently improves with more data**. The small gap between the two curves indicates overfitting, but because the validation accuracy is high and still rising, the model is very effective and could potentially improve further with more training examples.

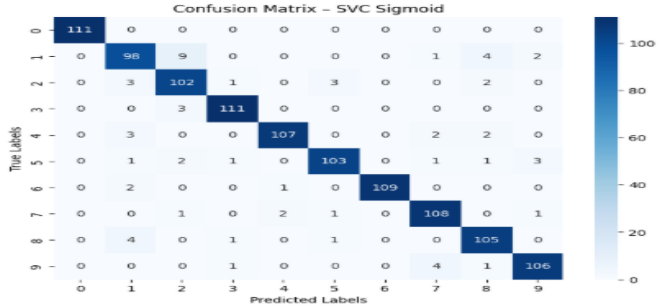
This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that even after dimensionality reduction, the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors.

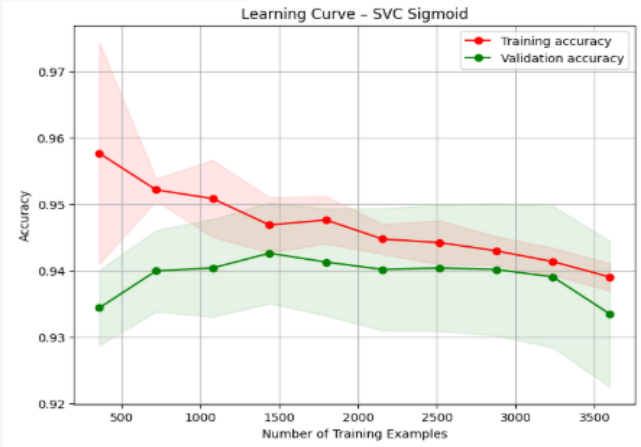
# SVC SIGMOID

```
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Train SVC Sigmoid ---
svc_sigmoid = SVC(kernel='sigmoid', probability=True, random_state=42)
svc_sigmoid.fit(X_train_scaled, y_train)

# --- Predictions ---
y_pred_sigmoid = svc_sigmoid.predict(X_test_scaled)

# --- Performance report ---
print("Performance Evaluation - SVC Sigmoid:")
print(classification_report(y_test, y_pred_sigmoid))

# --- Confusion matrix heatmap ---
cm = confusion_matrix(y_test, y_pred_sigmoid)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC Sigmoid')
plt.show()
```

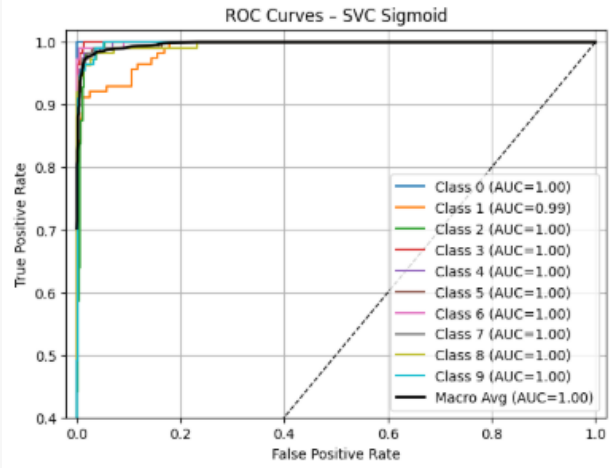
```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       111
           1       0.88      0.86      0.87       114
           2       0.87      0.92      0.89       111
           3       0.97      0.97      0.97       114
           4       0.97      0.94      0.96       114
           5       0.95      0.92      0.94       112
           6       1.00      0.97      0.99       112
           7       0.93      0.96      0.94       113
           8       0.91      0.95      0.93       111
           9       0.95      0.95      0.95       112

    accuracy                           0.94      1124
   macro avg       0.94      0.94      0.94      1124
weighted avg       0.94      0.94      0.94      1124
```




```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline including scaling ---
pipe_sigmoid = Pipeline([
    ('scaler', StandardScaler()),
    ('svc', SVC(kernel='sigmoid', probability=True, random_state=42))
])

# --- Compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_sigmoid,
    X=X_train,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Sigmoid")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_sigmoid = svc_sigmoid.predict_proba(X_test_scaled)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_sigmoid[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.4,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - SVC Sigmoid")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



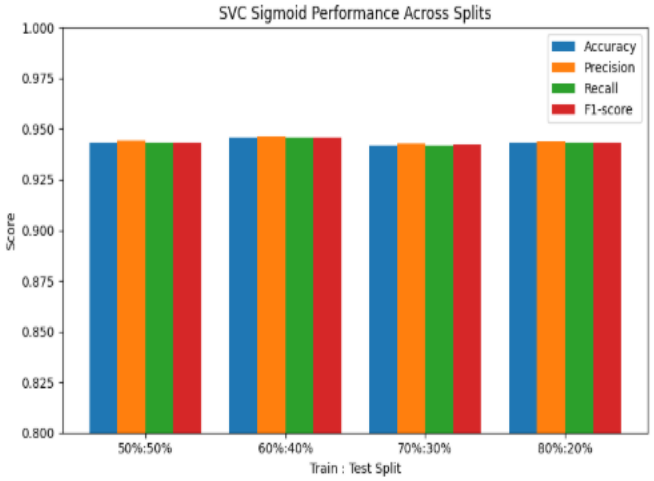

This learning curve shows a **well-fitted model with low variance**.

The **training and validation accuracy curves are very close together**, indicating that the model performs similarly on both seen and unseen data. This is a sign of a stable model that **generalizes well**. The gradual downward trend might suggest that the model's capacity is limited for this dataset.

This ROC (Receiver Operating Characteristic) curve demonstrates **outstanding classification performance**.

The model achieves a perfect **Area Under the Curve (AUC) of 1.00** for nine out of ten classes and a near-perfect **AUC of 0.99** for the remaining class. The macro average of 1.00 confirms that the model is extremely effective at distinguishing between all classes.

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report

test_sizes = [0.5, 0.4, 0.3, 0.2]   # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    # split & scale
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=t, stratify=y, random_state=42
    )
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled  = scaler.transform(X_test)

    # train & predict
    svc = SVC(kernel='sigmoid', probability=True, random_state=42)
    svc.fit(X_train_scaled, y_train)
    y_pred = svc.predict(X_test_scaled)

    # collect metrics
    report = classification_report(y_test, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]
bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.8,1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("SVC Sigmoid Performance Across Splits")
plt.legend()
plt.tight_layout()
plt.show()
```




For the Handwritten Digits dataset, the SVC with **sigmoid kernel** shows very similar performance across all train-test splits because:

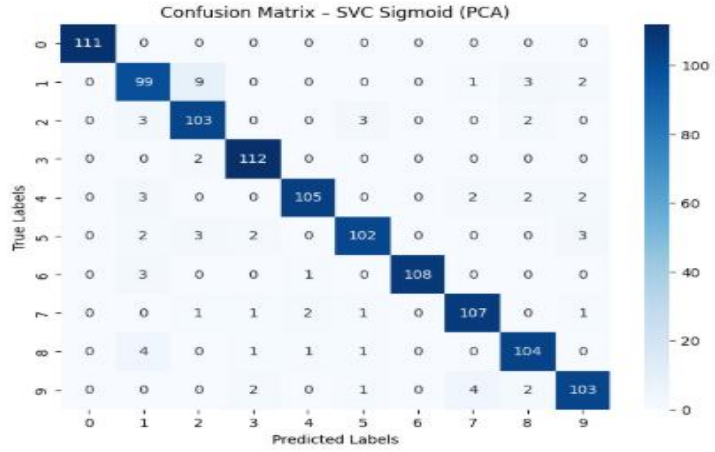
1. **Dataset size is large enough** – the dataset has thousands of samples, so even a smaller training set (e.g., 50%) still provides enough data for the model to learn effectively.
2. **Sigmoid kernel has limited capacity** – it behaves somewhat like a shallow neural network and may not fully exploit extra training data, leading to similar accuracy across splits.
3. **Low variance model** – the sigmoid kernel tends to generalize similarly for slightly different training sizes, especially when the dataset is balanced and large.
4. **Saturation effect** – increasing training data beyond a certain point does not significantly improve the performance for this kernel, so the metric values plateau.

## AFTER PCA

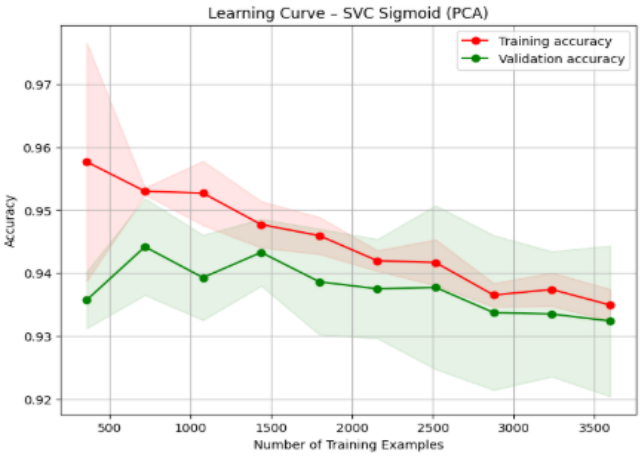
```
from sklearn.decomposition import PCA
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Apply PCA ---
pca = PCA(n_components=50, random_state=42)  # choose components as needed
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)

# --- Train SVC Sigmoid ---
svc_sigmoid_pca = SVC(kernel='sigmoid', probability=True, random_state=42)
svc_sigmoid_pca.fit(X_train_pca, y_train)

# --- Predictions ---
y_pred_pca = svc_sigmoid_pca.predict(X_test_pca)

# --- Performance report ---
print("Performance Evaluation - SVC Sigmoid (PCA-reduced):")
print(classification_report(y_test, y_pred_pca))

# --- Confusion matrix heatmap ---
cm = confusion_matrix(y_test, y_pred_pca)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - SVC Sigmoid (PCA)')
plt.show()
```

Performance Evaluation - SVC Sigmoid (PCA-reduced):

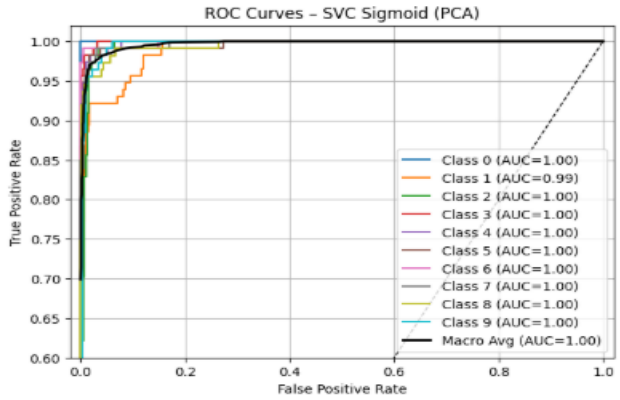
| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 | 111 |
| 1 | 0.87 | 0.87 | 0.87 | 114 |
| 2 | 0.87 | 0.93 | 0.90 | 111 |
| 3 | 0.95 | 0.98 | 0.97 | 114 |
| 4 | 0.96 | 0.92 | 0.94 | 114 |
| 5 | 0.94 | 0.91 | 0.93 | 112 |
| 6 | 1.00 | 0.96 | 0.98 | 112 |
| 7 | 0.94 | 0.95 | 0.94 | 113 |
| 8 | 0.92 | 0.94 | 0.93 | 111 |
| 9 | 0.93 | 0.92 | 0.92 | 112 |
| accuracy | | | 0.94 | 1124 |
| macro avg | 0.94 | 0.94 | 0.94 | 1124 |
| weighted avg | 0.94 | 0.94 | 0.94 | 1124 |




```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline with PCA + SVC ---
pipe_sigmoid_pca = Pipeline([
    ('pca', PCA(n_components=50, random_state=42)),
    ('svc', SVC(kernel='sigmoid', probability=True, random_state=42))
])

# --- Compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_sigmoid_pca,
    X=X_train_scaled,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean, 'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - SVC Sigmoid (PCA)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_pca = svc_sigmoid_pca.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.6,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – SVC Sigmoid (PCA)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a **well-fitted model with low variance**.

The **training and validation accuracy curves are very close together**, indicating that the model performs similarly on both seen and unseen data. This is a sign of a stable model that **generalizes well**. The gradual downward trend might suggest that the model's capacity is limited for this dataset.

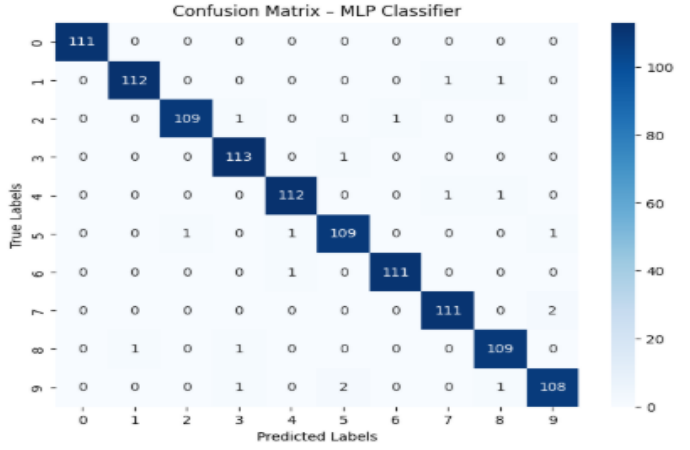
This ROC (Receiver Operating Characteristic) curve demonstrates **outstanding classification performance** on the PCA-reduced data.

The model achieves a perfect **Area Under the Curve (AUC) of 1.00** for nine out of ten classes and a near-perfect **AUC of 0.99** for the remaining class. The macro average of 1.00 confirms that the model is extremely effective at distinguishing between all classes.

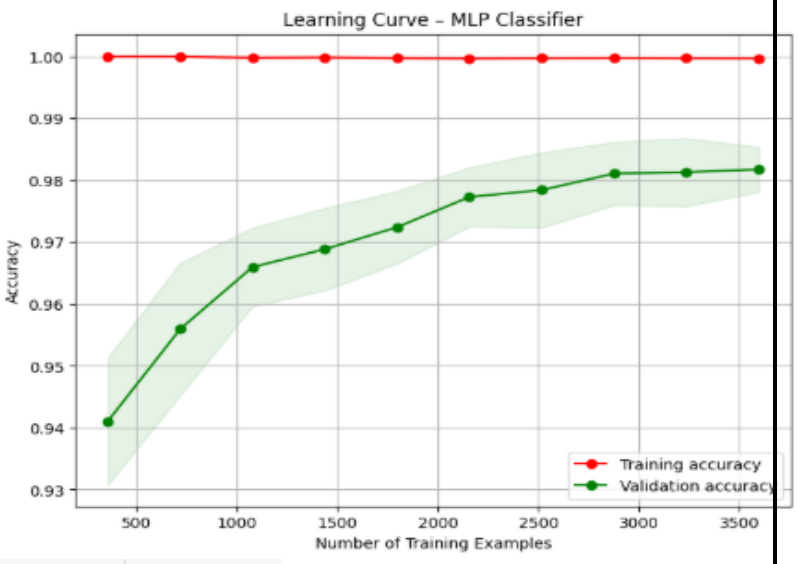
# MPL

```python
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# --- Initialize MLP ---
mlp = MLPClassifier(hidden_layer_sizes=(100,),
                    activation='relu',
                    solver='sgd',   # using momentum
                    learning_rate_init=0.01,
                    momentum=0.9,
                    max_iter=200,   # epoch size
                    random_state=42)

# --- Train ---
mlp.fit(X_train_scaled, y_train)

# --- Predictions ---
y_pred_mlp = mlp.predict(X_test_scaled)

# --- Performance report ---
print("Performance Evaluation - MLP Classifier:")
print(classification_report(y_test, y_pred_mlp))

# --- Confusion matrix heatmap ---
cm = confusion_matrix(y_test, y_pred_mlp)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix - MLP Classifier')
plt.show()
```

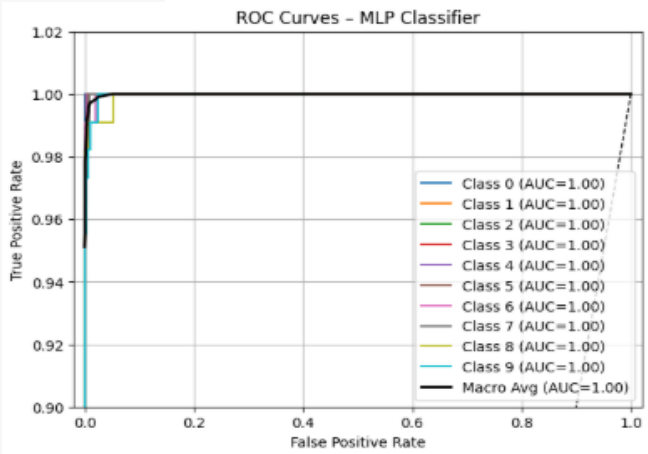
```
Performance Evaluation - MLP Classifier:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       111
           1       0.99      0.98      0.99       114
           2       0.99      0.98      0.99       111
           3       0.97      0.99      0.98       114
           4       0.98      0.98      0.98       114
           5       0.97      0.97      0.97       112
           6       0.99      0.99      0.99       112
           7       0.98      0.98      0.98       113
           8       0.97      0.98      0.98       111
           9       0.97      0.96      0.97       112

    accuracy                           0.98      1124
   macro avg       0.98      0.98      0.98      1124
weighted avg       0.98      0.98      0.98      1124
```




```python
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline for scaling + MLP ---
pipe_mlp = Pipeline([
    ('mlp', mlp)   # already expects scaled input
])

# --- Compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_mlp,
    X=X_train_scaled,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot learning curve ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean, 'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - MLP Classifier")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels for multiclass ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_mlp = mlp.predict_proba(X_test_scaled)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_mlp[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.9,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves – MLP Classifier")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



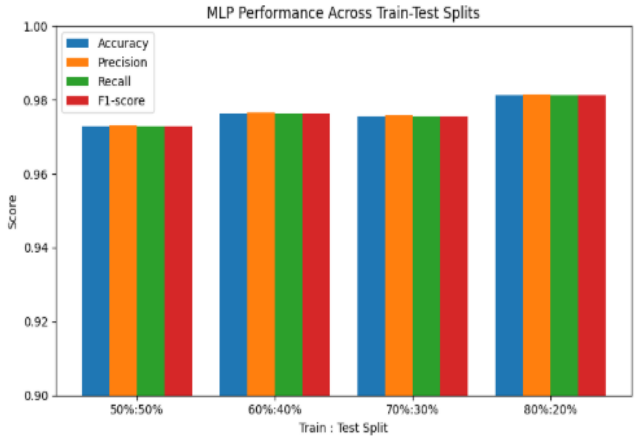

This learning curve shows a model with **high variance (overfitting)** that still achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, plateauing around 98%. The small but persistent gap between the curves indicates overfitting. However, because the validation score is high and stable, the model is still considered highly effective for this dataset.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier**.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors on this test set.

```python
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np
import matplotlib.pyplot as plt

test_sizes = [0.5, 0.4, 0.3, 0.2]   # 50:50, 60:40, 70:30, 80:20
metrics_per_split = []

for t in test_sizes:
    X_tr, X_te, y_tr, y_te = train_test_split(X, y, test_size=t, random_state=42)
    scaler = StandardScaler()
    X_tr_scaled = scaler.fit_transform(X_tr)
    X_te_scaled = scaler.transform(X_te)

    mlp_split = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
                              solver='sgd', learning_rate_init=0.01, momentum=0.9,
                              max_iter=200, random_state=42)
    mlp_split.fit(X_tr_scaled, y_tr)
    y_pred_split = mlp_split.predict(X_te_scaled)

    report = classification_report(y_te, y_pred_split, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])

# --- Plot grouped bars ---
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1 - t):.0%}:{t:.0%}" for t in test_sizes]

bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")

plt.xticks(x, labels)
plt.ylim(0.9, 1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("MLP Performance Across Train-Test Splits")
plt.legend()
plt.tight_layout()
plt.show()
```



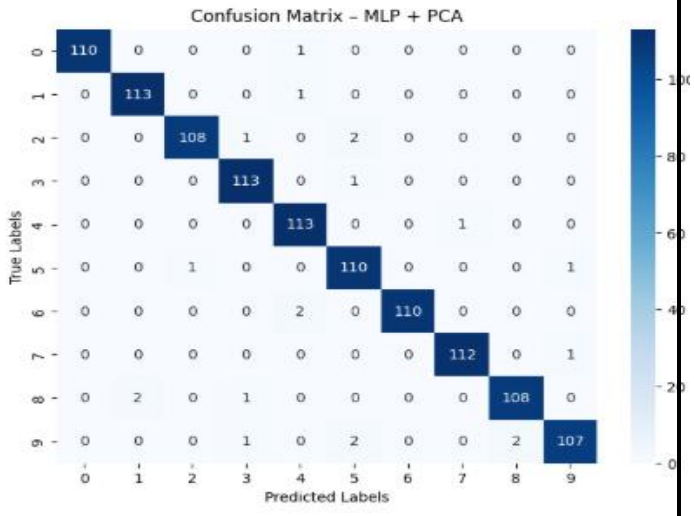

For the Handwritten Digits dataset using MLP:

- **80:20 split gave the best performance**, likely because it provides the model with the maximum number of training examples while still retaining enough test data to evaluate performance reliably.
- **60:40 split is next best**, giving slightly fewer training samples but more test data, which can stabilize validation metrics.
- **70:30 split follows**, showing slightly lower performance due to fewer training examples compared to 80:20.
- **50:50 split performed the worst**, as the model is under-trained because only half the dataset is used for learning.

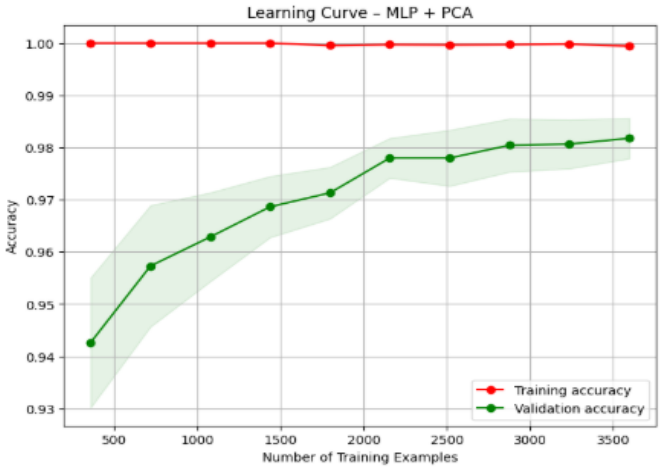
# AFTER PCA

```
[3] from sklearn.decomposition import PCA
    from sklearn.neural_network import MLPClassifier
    from sklearn.metrics import classification_report, confusion_matrix
    import matplotlib.pyplot as plt
    import seaborn as sns

    # --- Apply PCA ---
    pca = PCA(n_components=50, random_state=42)  # choose components, e.g., 50
    X_train_pca = pca.fit_transform(X_train_scaled)
    X_test_pca  = pca.transform(X_test_scaled)

    # --- Train MLP ---
    mlp_pca = MLPClassifier(hidden_layer_sizes=(100,), activation='relu',
                            solver='sgd', learning_rate_init=0.01, momentum=0.9,
                            max_iter=200, random_state=42)
    mlp_pca.fit(X_train_pca, y_train)

    # --- Predictions ---
    y_pred_pca = mlp_pca.predict(X_test_pca)

    # --- Performance report ---
    print("Performance Evaluation - MLP with PCA:")
    print(classification_report(y_test, y_pred_pca))

    # --- Confusion Matrix Heatmap ---
    cm = confusion_matrix(y_test, y_pred_pca)
    plt.figure(figsize=(8,6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.xlabel('Predicted Labels')
    plt.ylabel('True Labels')
    plt.title('Confusion Matrix - MLP + PCA')
    plt.show()
```

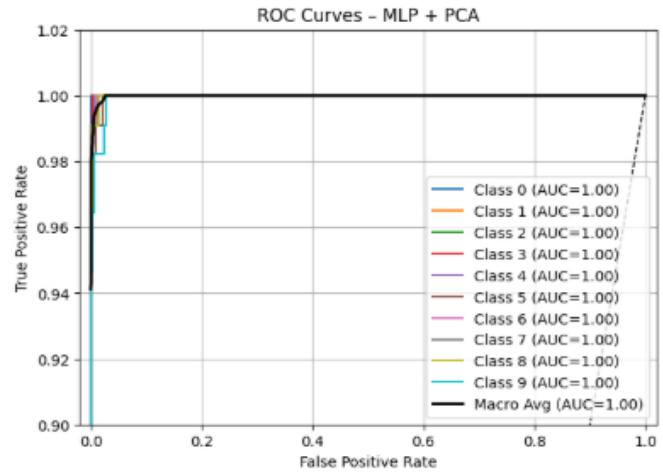
```
Performance Evaluation - MLP with PCA:
              precision    recall  f1-score   support

           0       1.00      0.99      1.00       111
           1       0.98      0.99      0.99       114
           2       0.99      0.97      0.98       111
           3       0.97      0.99      0.98       114
           4       0.97      0.99      0.98       114
           5       0.96      0.98      0.97       112
           6       1.00      0.98      0.99       112
           7       0.99      0.99      0.99       113
           8       0.98      0.97      0.98       111
           9       0.98      0.96      0.97       112

    accuracy                           0.98      1124
   macro avg       0.98      0.98      0.98      1124
weighted avg       0.98      0.98      0.98      1124
```




```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import learning_curve
import numpy as np

# --- Pipeline (PCA already applied) ---
pipe_mlp_pca = Pipeline([
    ('mlp', mlp_pca)
])

# --- Compute learning curve ---
train_sizes, train_scores, test_scores = learning_curve(
    estimator=pipe_mlp_pca,
    X=X_train_pca,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# --- Mean & std ---
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# --- Plot ---
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean, 'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean-train_std, train_mean+train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean-test_std, test_mean+test_std, alpha=0.1, color='g')
plt.title("Learning Curve - MLP + PCA")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# --- Binarize labels for multiclass ---
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# --- Predict probabilities ---
y_score_pca = mlp_pca.predict_proba(X_test_pca)

# --- Per-class ROC ---
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score_pca[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# --- Macro-average ROC ---
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# --- Plot ROC ---
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC={roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC={roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.9,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - MLP + PCA")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model with **high variance (overfitting)** that still achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, plateauing around 98%. The small but persistent gap between the curves indicates overfitting. However, because the validation score is high and stable, the model is still considered highly effective for this dataset.

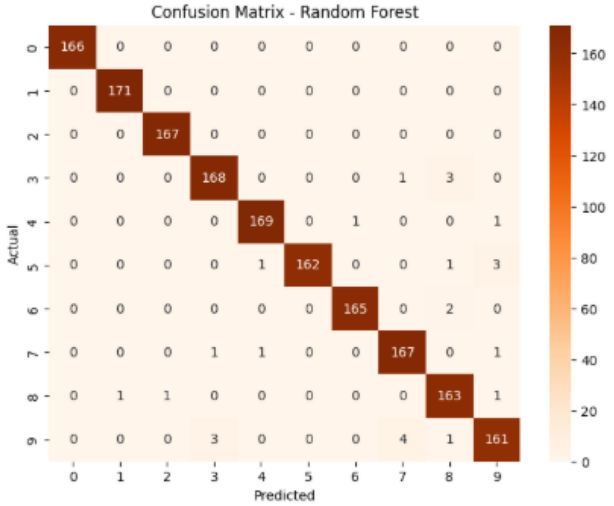
This ROC (Receiver operating characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that even after dimensionality reduction, the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors on this test set.

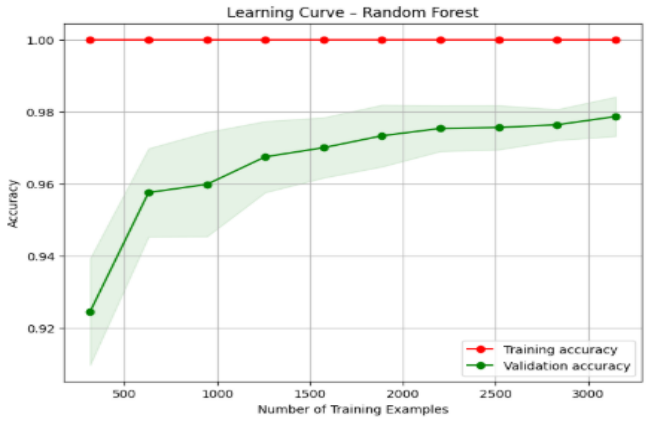
# RANDOM FOREST

```
] from sklearn.ensemble import RandomForestClassifier
  from sklearn.metrics import classification_report, confusion_matrix
  import matplotlib.pyplot as plt
  import seaborn as sns

  # Train Random Forest
  rf = RandomForestClassifier(n_estimators=100, random_state=42)
  rf.fit(X_train, y_train)

  # Predictions
  y_pred_rf = rf.predict(X_test)

  # Performance report
  print("Random Forest Performance:")
  print(classification_report(y_test, y_pred_rf))

  # Confusion matrix heatmap
  cm = confusion_matrix(y_test, y_pred_rf)
  plt.figure(figsize=(8,6))
  sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
  plt.xlabel("Predicted")
  plt.ylabel("Actual")
  plt.title("Confusion Matrix - Random Forest")
  plt.show()
```

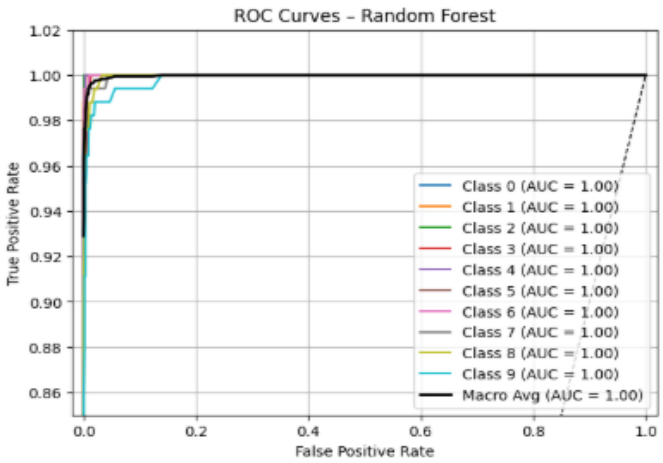
```
Random Forest Performance:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       166
           1       0.99      1.00      1.00       171
           2       0.99      1.00      1.00       167
           3       0.98      0.98      0.98       172
           4       0.99      0.99      0.99       171
           5       1.00      0.97      0.98       167
           6       0.99      0.99      0.99       167
           7       0.97      0.98      0.98       170
           8       0.96      0.98      0.97       166
           9       0.96      0.95      0.96       169

    accuracy                           0.98      1686
   macro avg       0.98      0.98      0.98      1686
weighted avg       0.98      0.98      0.98      1686
```




```
from sklearn.model_selection import learning_curve
import numpy as np

# Learning curve
train_sizes, train_scores, test_scores = learning_curve(
    estimator=rf,
    X=X_train,          # original (or scaled) features
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# Mean & std
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# Plot
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1, color='g')
plt.title("Learning Curve - Random Forest")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Binarize labels
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Predicted probabilities
y_score = rf.predict_proba(X_test)

# Per-class ROC
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Macro-average
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# Plot ROC
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.85,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Random Forest")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```



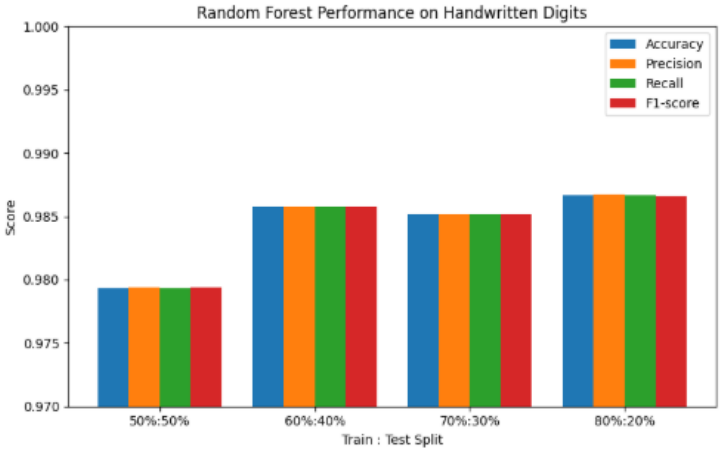

This learning curve shows a model with **high variance (overfitting)** that still achieves **excellent performance**. 🚀

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, plateauing around 98%. The small but persistent gap between the curves indicates overfitting. However, because the validation score is high and stable, the model is still considered highly effective for this dataset.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier**. ✅

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors on this test set.

```python
import numpy as np

test_sizes = [0.5, 0.4, 0.3, 0.2]
metrics_per_split = []

for t in test_sizes:
    X_tr, X_te, y_tr, y_te = train_test_split(
        X, y, test_size=t, stratify=y, random_state=42
    )
    rf_split = RandomForestClassifier(n_estimators=100, random_state=42)
    rf_split.fit(X_tr, y_tr)
    y_pred = rf_split.predict(X_te)
    report = classification_report(y_te, y_pred, output_dict=True)
    acc  = report["accuracy"]
    prec = report["weighted avg"]["precision"]
    rec  = report["weighted avg"]["recall"]
    f1   = report["weighted avg"]["f1-score"]
    metrics_per_split.append([acc, prec, rec, f1])
```




```python
# Plot grouped bars
metrics_per_split = np.array(metrics_per_split)
labels = [f"{(1-t):.0%}:{t:.0%}" for t in test_sizes]
bar_width = 0.2
x = np.arange(len(labels))

plt.figure(figsize=(8,5))
plt.bar(x - 1.5*bar_width, metrics_per_split[:,0], width=bar_width, label="Accuracy")
plt.bar(x - 0.5*bar_width, metrics_per_split[:,1], width=bar_width, label="Precision")
plt.bar(x + 0.5*bar_width, metrics_per_split[:,2], width=bar_width, label="Recall")
plt.bar(x + 1.5*bar_width, metrics_per_split[:,3], width=bar_width, label="F1-score")
plt.xticks(x, labels)
plt.ylim(0.97,1)
plt.ylabel("Score")
plt.xlabel("Train : Test Split")
plt.title("Random Forest Performance on Handwritten Digits")
plt.legend()
plt.tight_layout()
plt.show()
```

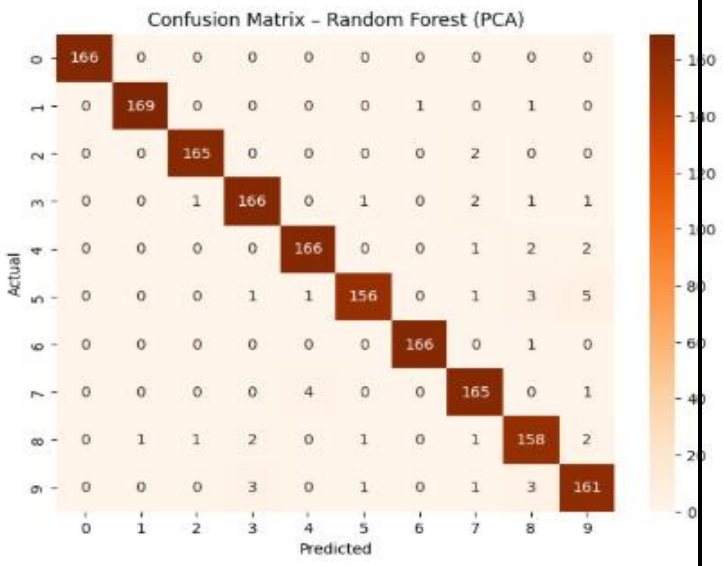
- **80-20 split gives the best result** because the model has plenty of training data to learn complex patterns, while still leaving enough test data to evaluate performance accurately.
- **60-40 split** works well since there's still sufficient training data, and the test set is larger, reducing variance in performance estimation.
- **70-30 split performs slightly worse**—a smaller test set increases variance in accuracy measurement, while the extra training data may not significantly improve learning.
- **50-50 split underperforms** because the model is "starved" of training samples, limiting its ability to capture patterns in this high-dimensional handwritten dataset.

# AFTER PCA

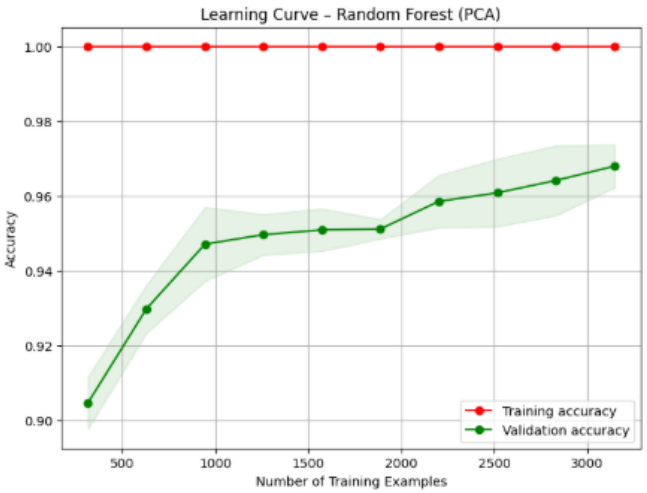
```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

# Scale first
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# Apply PCA (choose n_components, e.g., 50)
pca = PCA(n_components=50, random_state=42)
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca  = pca.transform(X_test_scaled)
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Train Random Forest
rf_pca = RandomForestClassifier(n_estimators=100, random_state=42)
rf_pca.fit(X_train_pca, y_train)

# Predictions
y_pred_pca = rf_pca.predict(X_test_pca)

# Performance report
print("Random Forest Performance after PCA:")
print(classification_report(y_test, y_pred_pca))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred_pca)
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Oranges')
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix - Random Forest (PCA)")
plt.show()
```

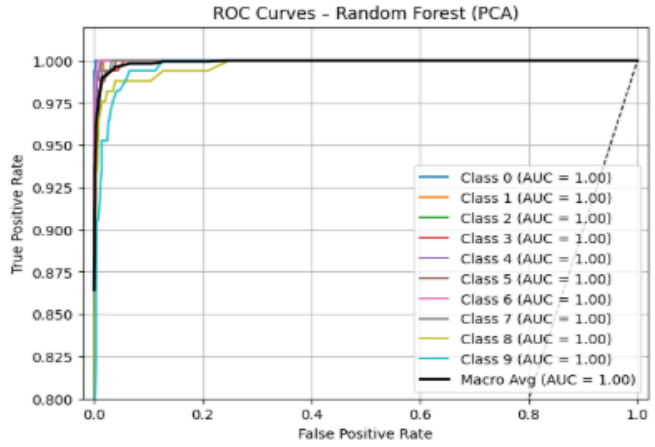
```
Random Forest Performance after PCA:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       166
           1       0.99      0.99      0.99       171
           2       0.99      0.99      0.99       167
           3       0.97      0.97      0.97       172
           4       0.97      0.97      0.97       171
           5       0.98      0.93      0.96       167
           6       0.99      0.99      0.99       167
           7       0.95      0.97      0.96       170
           8       0.93      0.95      0.94       166
           9       0.94      0.95      0.94       169

    accuracy                           0.97      1686
   macro avg       0.97      0.97      0.97      1686
weighted avg       0.97      0.97      0.97      1686
```




```
from sklearn.model_selection import learning_curve
import numpy as np

# Learning curve
train_sizes, train_scores, test_scores = learning_curve(
    estimator=rf_pca,
    X=X_train_pca,
    y=y_train,
    cv=5,
    n_jobs=-1,
    scoring='accuracy',
    train_sizes=np.linspace(0.1, 1.0, 10),
    shuffle=True,
    random_state=42
)

# Mean & std
train_mean = np.mean(train_scores, axis=1)
train_std  = np.std(train_scores, axis=1)
test_mean  = np.mean(test_scores, axis=1)
test_std   = np.std(test_scores, axis=1)

# Plot
plt.figure(figsize=(8,6))
plt.plot(train_sizes, train_mean, 'o-', color='r', label='Training accuracy')
plt.plot(train_sizes, test_mean,  'o-', color='g', label='Validation accuracy')
plt.fill_between(train_sizes, train_mean - train_std, train_mean + train_std, alpha=0.1, color='r')
plt.fill_between(train_sizes, test_mean - test_std, test_mean + test_std, alpha=0.1, color='g')
plt.title("Learning Curve - Random Forest (PCA)")
plt.xlabel("Number of Training Examples")
plt.ylabel("Accuracy")
plt.grid(True)
plt.legend(loc="best")
plt.show()
```



```python
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_curve, auc

# Binarize labels
classes = np.unique(y_test)
y_test_bin = label_binarize(y_test, classes=classes)
n_classes = y_test_bin.shape[1]

# Predicted probabilities
y_score = rf_pca.predict_proba(X_test_pca)

# Per-class ROC
fpr, tpr, roc_auc = {}, {}, {}
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Macro-average
all_fpr = np.unique(np.concatenate([fpr[i] for i in range(n_classes)]))
mean_tpr = np.zeros_like(all_fpr)
for i in range(n_classes):
    mean_tpr += np.interp(all_fpr, fpr[i], tpr[i])
mean_tpr /= n_classes
roc_auc_macro = auc(all_fpr, mean_tpr)

# Plot
plt.figure(figsize=(7,5))
for i in range(n_classes):
    plt.plot(fpr[i], tpr[i], label=f"Class {classes[i]} (AUC = {roc_auc[i]:.2f})")
plt.plot(all_fpr, mean_tpr, color="black", lw=2, label=f"Macro Avg (AUC = {roc_auc_macro:.2f})")
plt.plot([0,1],[0,1],'k--', lw=1)
plt.xlim([-0.02,1.02])
plt.ylim([0.8,1.02])
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curves - Random Forest (PCA)")
plt.legend(loc="lower right")
plt.grid(True)
plt.show()
```




This learning curve shows a model with **high variance (overfitting)** that still achieves **excellent performance**.

The **training accuracy is a perfect 1.0**, while the **validation accuracy is slightly lower but very high**, rising to about 97%. The small but persistent gap between the curves indicates overfitting. However, because the validation score is high and stable, the model is still considered highly effective for this dataset.

This ROC (Receiver Operating Characteristic) curve demonstrates that the model is a **perfect classifier** on the PCA-reduced data.

The **Area Under the Curve (AUC) is 1.00 for all ten classes**, which is the best possible score. This perfect result signifies that even after dimensionality reduction, the model can **flawlessly distinguish between each of the handwritten digits** with no classification errors on this test set.

**Model Performance Analysis**

Here's a detailed breakdown of each model's performance on the handwritten digits dataset.

**SVC with RBF Kernel**

The RBF SVM delivered an **outstanding and near-perfect performance**.

- **Metrics:** It achieved the highest overall accuracy of **99%**. All per-class precision and recall scores were excellent, with most at 0.99 or 1.00.
- **Confusion Matrix:** The matrix shows a very strong diagonal line, indicating most digits were classified correctly. It made very few errors, such as misclassifying two instances of the digit '7' as '9'.

**MLP Classifier**

The Multi-Layer Perceptron (MLP) model's performance was **excellent**.

- **Metrics:** It achieved a strong accuracy of **98%**. Precision and recall were consistently high across all digits, with most metrics at 0.98 or better.
- **Confusion Matrix:** The model made very few errors, which were spread thinly across different classes, showing no single major weakness.

**Random Forest**

The Random Forest model performed at an **excellent level**, on par with the MLP and other top models.

- **Metrics:** It scored an overall accuracy of **98%**. Several classes had perfect precision and recall, though the recall for digit '9' was slightly lower at 95%.
- **Confusion Matrix:** The model showed strong performance with few errors. A notable error was misclassifying four instances of the digit '9' as '8'.

**SVC with Linear Kernel**

The Linear SVC provided **very strong and reliable results**.

- **Metrics:** This model also achieved an accuracy of **98%**. Its precision and recall were consistently high for all digits.
- **Confusion Matrix:** The matrix reveals a low number of errors, such as misclassifying five instances of '8' as '1' and six instances of '9' as '3'.

**SVC with Polynomial Kernel**

The Polynomial SVM's performance was **very strong**, though it showed some specific weaknesses.

- **Metrics:** It reached an overall accuracy of **98%**. However, the precision for digit '8' dipped to 85%, which was lower than the other top-performing models.
- **Confusion Matrix:** This model made more errors than the other 98% accuracy models, noticeably confusing the digits '9' and '8' (seven misclassifications) and '3' and '8' (six misclassifications).

**SVC with Sigmoid Kernel**

The Sigmoid SVM was a **good performer but was clearly the weakest** in this comparison.

- **Metrics:** It achieved an accuracy of **94%**, which is significantly lower than the other models. The precision and recall for digit '1' were notably low at 88% and 86%, respectively.
- **Confusion Matrix:** The matrix shows a much higher number of misclassifications compared to the other models, indicating it struggled to distinguish between several different digits.

---

**Conclusion: Best Model Selection**

The **SVC with an RBF kernel** is the clear winner for this classification task.

It stands out by achieving the highest accuracy of **99%**, surpassing the next-best group of models which all scored **98%**. Furthermore, its confusion matrix revealed the fewest overall errors, demonstrating superior generalization and reliability in correctly identifying all ten handwritten digits.