Digit Recognition using PCA + kNN

Name - Sonalika Chandra
Roll No. - 2201CS68
Course - Advanced Pattern Recognition

Introduction

Pattern recognition plays a central role in machine learning, image processing, and computer vision. In this project, we implement a **digit recognition system** using two fundamental techniques:

1. **Principal Component Analysis (PCA):** A dimensionality reduction method that projects high-dimensional data into a lower-dimensional subspace while retaining maximum variance.

2. **k-Nearest Neighbors (kNN):** A non-parametric classifier that assigns labels based on the majority class among the nearest neighbors in feature space.

The combination of PCA and kNN provides a computationally efficient and interpretable model that works well for image classification tasks.

For evaluation, we use the **Digits dataset** from `scikit-learn` (8×8 grayscale digit images) and the **MNIST dataset** (28×28 handwritten digits).

Objectives

- To apply PCA for feature extraction and dimensionality reduction.
- To classify digits using the kNN algorithm.
- To evaluate performance using accuracy, classification report, and confusion matrix.
- To visualize results with PCA variance plots and eigen-digits.

Methodology

- Step 1: Data loading (digits / MNIST).
- Step 2: Preprocessing (scaling).
- Step 3: Apply PCA (retain 95% variance).
- Step 4: Train kNN.
- Step 5: Evaluate (accuracy, confusion matrix).

Implementation

- numpy for numerical computation.
- scikit-learn for PCA, kNN, and evaluation metrics.
- matplotlib for visualization.

Code

USING DIGITS DATASET(8×8)

Load dataset

```
USE_MNIST = False  # False = sklearn.load_digits (fast), True = MNIST
(large, downloads ~50MB)
#USE_MNIST = True #(Use it to downloads ~50MB from OpenML )

if not USE_MNIST:
    digits = load_digits()
    X, y = digits.data, digits.target
    image_shape = (8, 8)
    print("Dataset: sklearn.load_digits", X.shape)
else:
    mnist = fetch_openml('mnist_784', version=1, as_frame=False)
    X, y = mnist.data.astype(np.float32), mnist.target.astype(int)
    image_shape = (28, 28)
    print("Dataset: MNIST", X.shape)
```

Output →

```
Dataset: sklearn.load_digits (1797, 64)
```

Train-Test Split

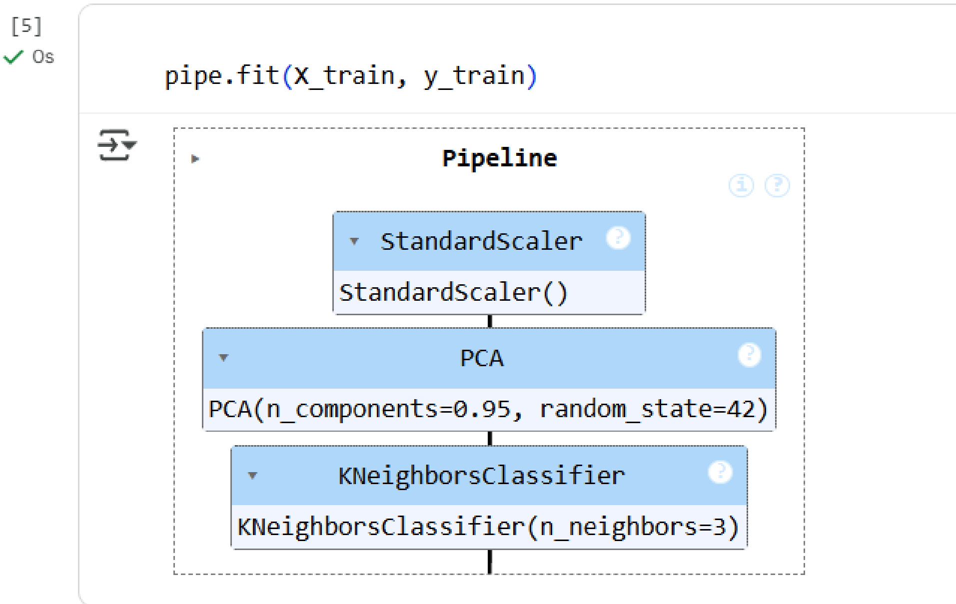
```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

Pipeline: Scale → PCA → kNN

```
pipe = Pipeline([
    ("scaler", StandardScaler()),
    ("pca", PCA(n_components=0.95, random_state=42)),  # keep 95% variance
    ("knn", KNeighborsClassifier(n_neighbors=3))
])
```

Train

```
pipe.fit(X_train, y_train)
```



Output →

Predict

```
y_pred = pipe.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {acc:.4f}")
print(classification_report(y_test, y_pred))
```

Output →

```
[6]
0s
y_pred = pipe.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {acc:.4f}")
print(classification_report(y_test, y_pred))
```

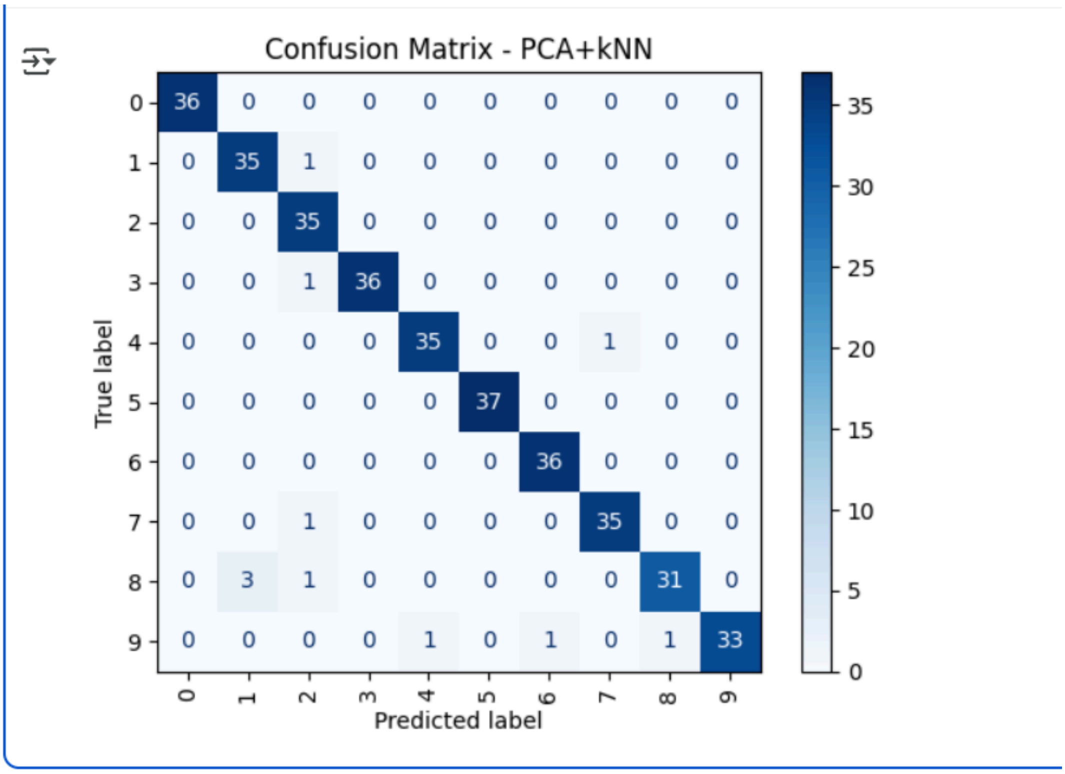
```
Test Accuracy: 0.9694
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        36
           1       0.92      0.97      0.95        36
           2       0.90      1.00      0.95        35
           3       1.00      0.97      0.99        37
           4       0.97      0.97      0.97        36
           5       1.00      1.00      1.00        37
           6       0.97      1.00      0.99        36
           7       0.97      0.97      0.97        36
           8       0.97      0.89      0.93        35
           9       1.00      0.92      0.96        36

    accuracy                           0.97       360
   macro avg       0.97      0.97      0.97       360
weighted avg       0.97      0.97      0.97       360
```

Confusion Matrix

```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot(cmap="Blues", xticks_rotation='vertical')
plt.title("Confusion Matrix - PCA+kNN")
plt.show()
```

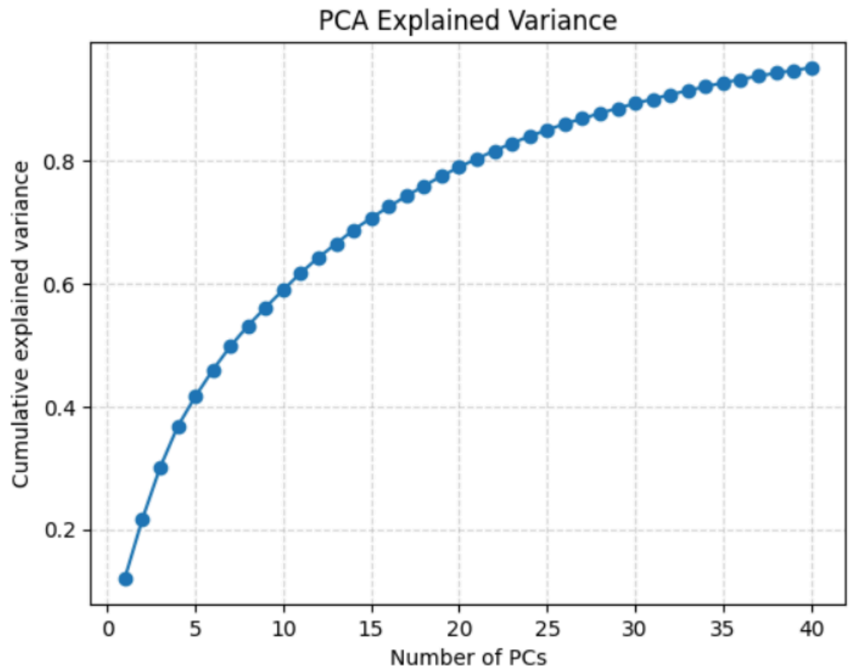


Output →

Explained Variance

```
pca_model = pipe.named_steps["pca"]
cumsum = np.cumsum(pca_model.explained_variance_ratio_)
plt.plot(np.arange(1, len(cumsum)+1), cumsum, marker="o")
plt.xlabel("Number of PCs")
plt.ylabel("Cumulative explained variance")
plt.title("PCA Explained Variance")
plt.grid(True, linestyle="--", alpha=0.5)
plt.show()
```

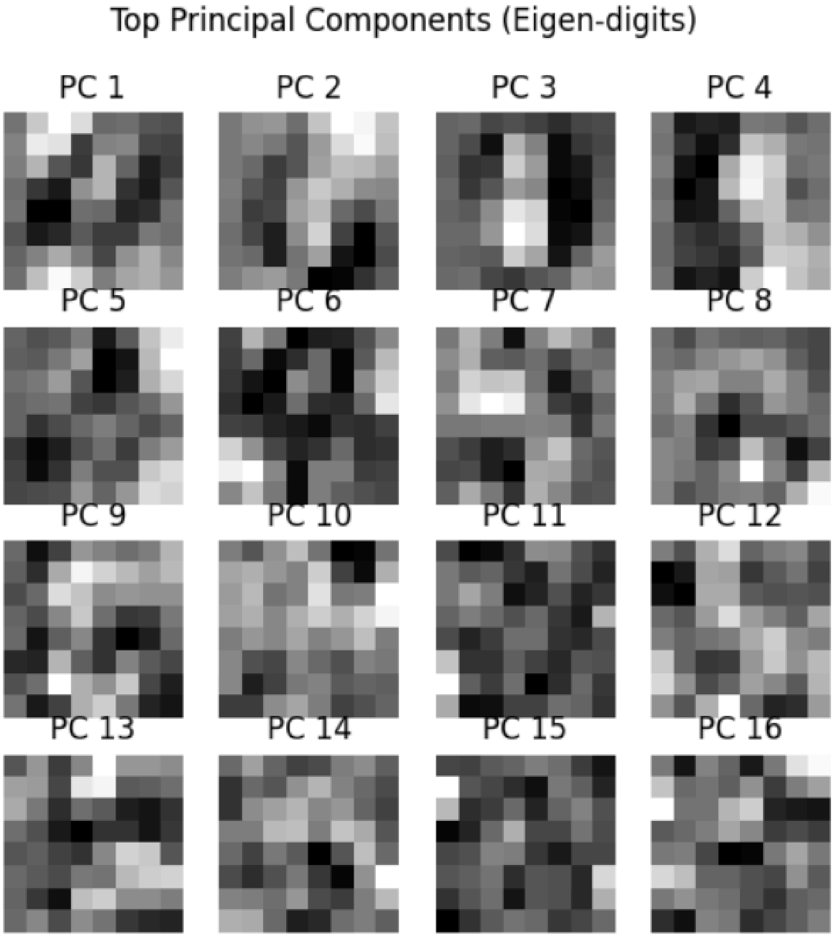
Output →



Visualize Top PCs

```
n_show = 16
fig, axes = plt.subplots(4, 4, figsize=(6, 6))
for i, ax in enumerate(axes.ravel()):
    if i < pca_model.components_.shape[0]:
        ax.imshow(pca_model.components_[i].reshape(image_shape),
cmap="gray")
        ax.set_title(f"PC {i+1}")
        ax.axis("off")
plt.suptitle("Top Principal Components (Eigen-digits)")
plt.show()
```

Output →

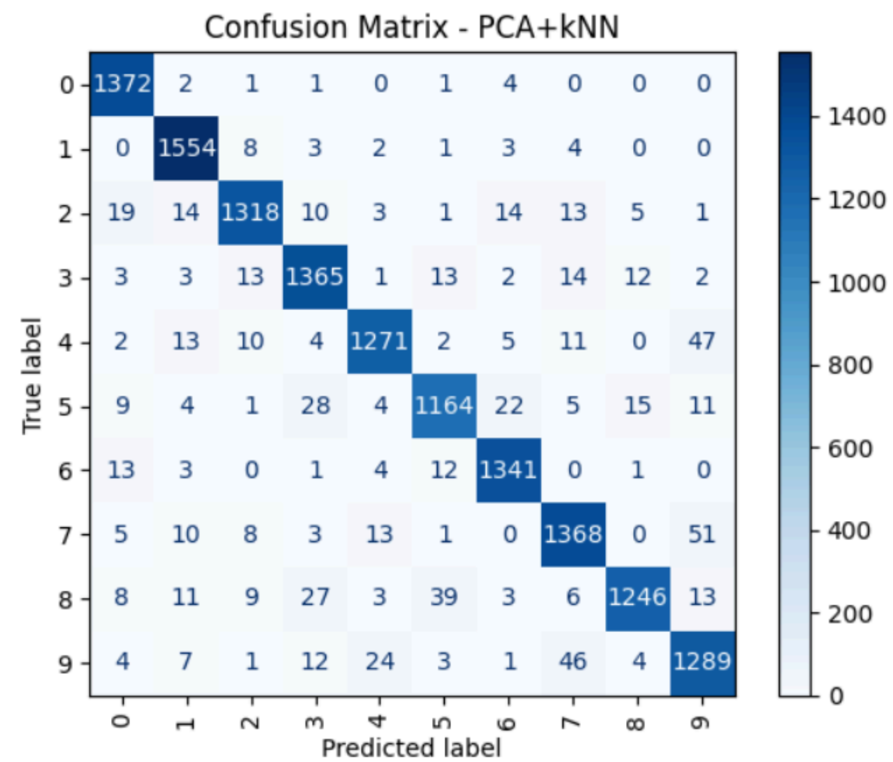


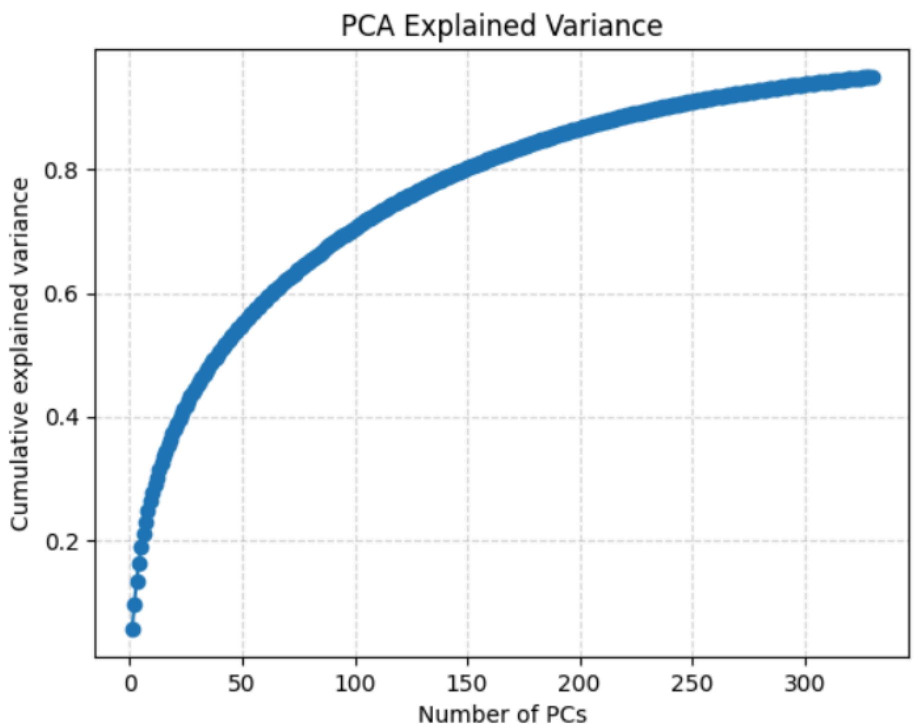
USING MNIST DATASET (28 x 28)

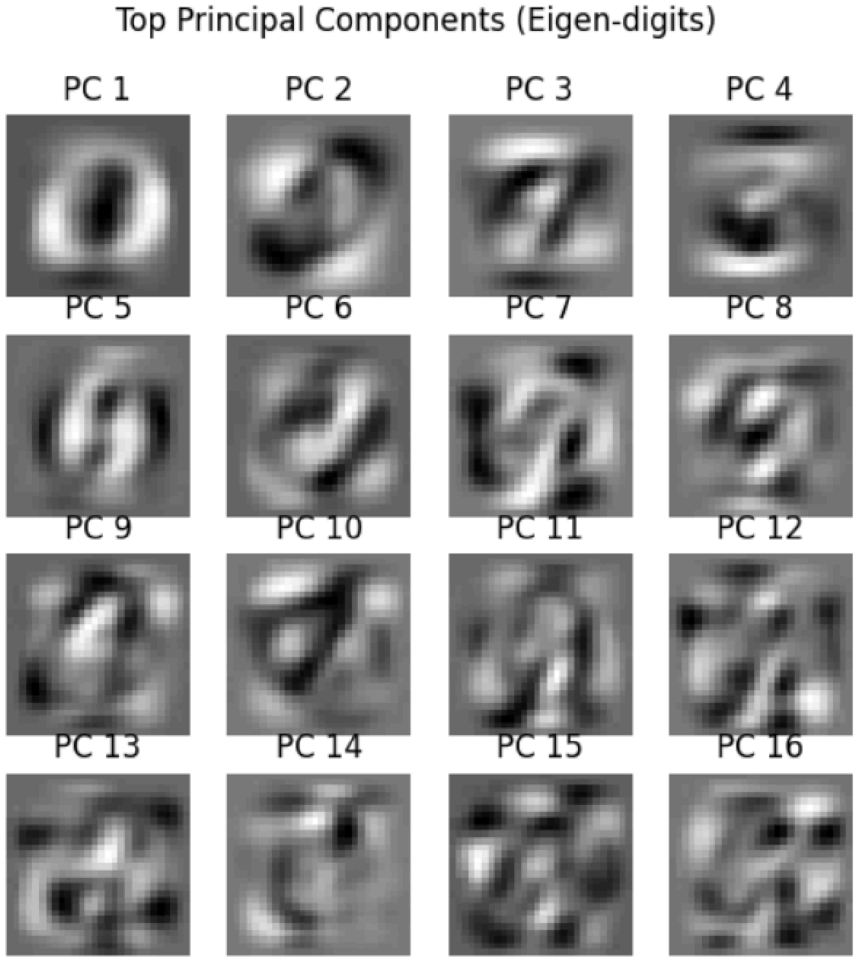
```
Test Accuracy: 0.9491
              precision    recall  f1-score   support

           0       0.96      0.99      0.97      1381
           1       0.96      0.99      0.97      1575
           2       0.96      0.94      0.95      1398
           3       0.94      0.96      0.95      1428
           4       0.96      0.93      0.94      1365
           5       0.94      0.92      0.93      1263
           6       0.96      0.98      0.97      1375
           7       0.93      0.94      0.94      1459
           8       0.97      0.91      0.94      1365
           9       0.91      0.93      0.92      1391

    accuracy                           0.95     14000
   macro avg       0.95      0.95      0.95     14000
weighted avg       0.95      0.95      0.95     14000
```


Confusion Matrix - PCA+kNN
True label
Predicted label
0 1 2 3 4 5 6 7 8 9
1372 2 1 1 0 1 4 0 0 0
0 1554 8 3 2 1 3 4 0 0
19 14 1318 10 3 1 14 13 5 1
3 3 13 1365 1 13 2 14 12 2
2 13 10 4 1271 2 5 11 0 47
9 4 1 28 4 1164 22 5 15 11
13 3 0 1 4 12 1341 0 1 0
5 10 8 3 13 1 0 1368 0 51
8 11 9 27 3 39 3 6 1246 13
4 7 1 12 24 3 1 46 4 1289
1400
1200
1000
800
600
400
200
0


PCA Explained Variance
Cumulative explained variance
0.8
0.6
0.4
0.2
0
50
100
150
200
250
300
Number of PCs


Top Principal Components (Eigen-digits)
PC 1
PC 2
PC 3
PC 4
PC 5
PC 6
PC 7
PC 8
PC 9
PC 10
PC 11
PC 12
PC 13
PC 14
PC 15
PC 16