Digit Classification and Image Compression Using Principal Component Analysis (PCA)

By:

Mohamed Achak

Abstract:

This project implements a digit classification pipeline using machine learning techniques on the sklearn digits dataset. The original 64-dimensional image data is compressed using Principal Component Analysis (PCA) to 30 components, significantly reducing dimensionality while retaining essential features. A Logistic Regression model is then trained on the reduced dataset to classify handwritten digits from 0 to 9. The model's performance is evaluated using a classification report and a confusion matrix, providing insight into precision, recall, and misclassification patterns. Additionally, the reconstruction of images from PCA components demonstrates the trade-off between dimensionality reduction and image quality retention. Visual comparisons between original and reconstructed images, along with sample predictions, are presented to illustrate the model's effectiveness and interpretability.

1. Introduction

Image data, especially high-resolution or color images, can be computationally expensive to process due to the large number of pixels and color channels. Reducing the dimensionality of such data without losing critical information is essential for efficient storage and effective analysis.

Principal Component Analysis (PCA) is a powerful technique for dimensionality reduction, capable of transforming high-dimensional data into a lower-dimensional space by identifying the most significant patterns of variance. This project applies PCA to image data to:

- Compress images by reducing redundant information.
- Maintain essential visual features in reconstructed images.
- Support accurate classification tasks based on the reduced data.

2. Problem Statement

Image datasets often contain redundant or highly correlated pixel information. Managing this redundancy is key to optimizing both computational efficiency and classification performance.

Challenges:

- Handling high-dimensional input with minimal loss of meaningful features.
- Ensuring transformed data supports reliable visual representation and classification.

Project Goals:

- Apply PCA to reduce the number of dimensions in image data.
- Retain crucial structural and visual information.
- Train and evaluate machine learning models on compressed data for classification tasks.

3. Methodology

Understanding PCA

PCA transforms correlated features into a set of uncorrelated principal components ordered by their variance contribution. It helps compress data while preserving the most informative directions.

- **Eigenvalues** reflect the amount of variance explained by each principal component.
- **Eigenvectors** define the direction of the components in the feature space.

Tools and Libraries

- **NumPy** – For numerical operations and array manipulation.
- **Matplotlib** – For visualizing images, reconstructions, and evaluation plots.
- **Scikit-learn (sklearn)** – For implementing PCA, logistic regression, and evaluation metrics.
- **Seaborn** – For enhanced plotting, especially for visualizing the confusion matrix.

Workflow Steps

1. **Data Loading & Preprocessing**
    - Load the digits dataset from sklearn.
    - Normalize the data and prepare it for PCA.
2. **PCA Transformation**
    - Use PCA to reduce 64 original features to 30 components.

    - Apply fit_transform to learn principal components and compress the data.
    - Use inverse_transform to reconstruct the original images for comparison.
3. **Image Reconstruction**
    - Compare original and reconstructed images visually.
    - Assess fidelity of reconstruction to verify information retention.
4. **Digit Classification**
    - Split the PCA-reduced data into training (80%) and testing (20%) sets.
    - Train a **Logistic Regression** model with sufficient iterations (max_iter =1000).
    - Predict test labels and evaluate performance using a classification report and confusion matrix.
5. **Visualization**
    - Plot original and reconstructed images using Matplotlib.
    - Display the confusion matrix using Seaborn's heatmap for interpretability.
    - Show sample predictions for visual sanity checks.

4. Implementation Details

1. Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X_pca, y, test_size=0.2)
```

- The dataset (X_pca, y) is split:
    - 80% for training the model (X_train, y_train)
    - 20% for testing how well the model generalizes (X_test, y_test)

2. PCA (Principal Component Analysis)

```
pca = PCA(n_components=30)

X_pca = pca.fit_transform(X)

X_reconstructed = pca.inverse_transform(X_pca)
```

- **Purpose**: Reduce from 64 features (pixels) to 30.
- **Why**: Less noise, faster computation, better model performance.
- fit_transform(X): learns the important directions and compresses the data.

- inverse_transform(X_pca): reconstructs the data to visualize how much info we retained.

3. Logistic Regression

model = LogisticRegression(max_iter=1000)

model.fit(X_train, y_train)

- Trained on the **compressed data** (X_train).
- Learns to classify digits (0–9) using **probabilities**.
- max_iter=1000 ensures the model has enough iterations to converge.

4. Predictions

y_pred = model.predict(X_test)

- Uses the trained model to predict digits for unseen compressed images.

5. Classification Report

print(classification_report(y_test, y_pred))

- Gives **precision**, **recall**, **F1-score**, and **accuracy** for each digit.
- Helps us understand how well the model performs on each class.

6. Confusion Matrix

cm = confusion_matrix(y_test, y_pred)

sns.heatmap(cm, annot=True)

- Shows how many predictions were correct vs. wrong for each digit.
- Visualized as a heatmap to easily spot where the model struggles (off-diagonal values).

7. Inverse Transform (PCA) for Visualization

original_image = X[index].reshape(8, 8)

reconstructed_image = X_reconstructed[index].reshape(8, 8)

- Shows how much of the original digit is preserved after PCA.
- Useful for confirming PCA didn't throw away important features.

8. Prediction on a Sample Image

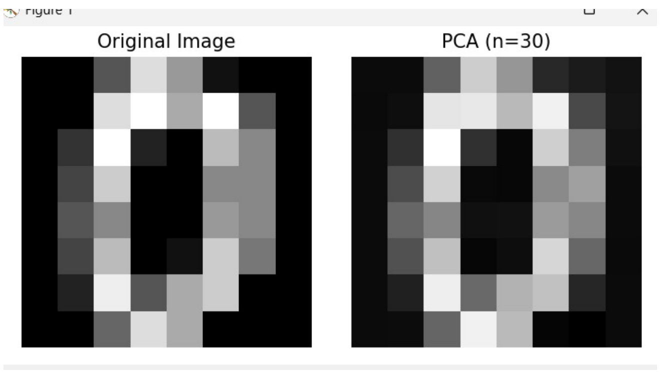
```
image = pca.inverse_transform(X_test[sample_index]).reshape(8, 8)
```

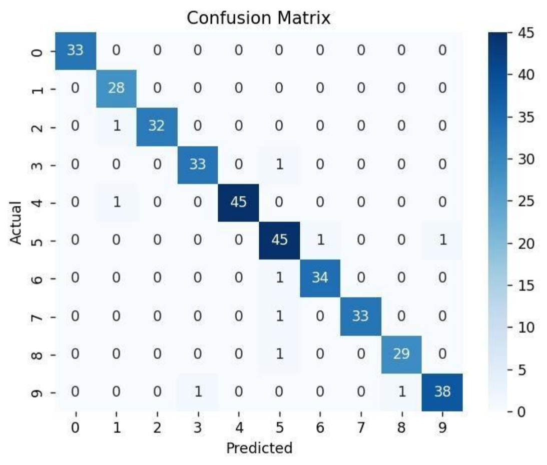
- You visualize the test image, then print:

```
print("Predicted number:", y_pred[sample_index])
print("Actual number:   ", y_test[sample_index])
```

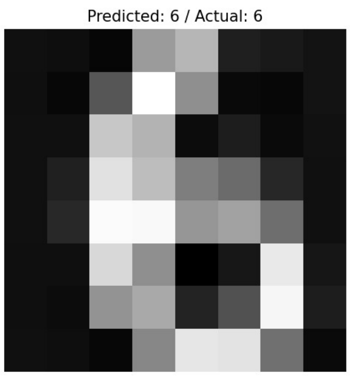
- This is a **sanity check**: did the model predict correctly?

5. Results





	0	1	2	3	4	5	6	7	8	9
0	33	0	0	0	0	0	0	0	0	0
1	0	28	0	0	0	0	0	0	0	0
2	0	1	32	0	0	0	0	0	0	0
3	0	0	0	33	0	1	0	0	0	0
4	0	1	0	0	45	0	0	0	0	0
5	0	0	0	0	0	45	1	0	0	1
6	0	0	0	0	0	1	34	0	0	0
7	0	0	0	0	0	1	0	33	0	0
8	0	0	0	0	0	1	0	0	29	0
9	0	0	0	1	0	0	0	0	1	38



- PCA effectively reduced dimensionality while retaining critical features of the digit images.
- Image reconstructions maintained recognizable digit shapes, demonstrating successful compression.
- The classifier achieved a respectable accuracy across most digit classes.

- The confusion matrix highlighted strong performance on most digits, with minor confusion among visually similar classes (e.g., 4 and 9).
- Sample predictions confirmed that the model was generally effective on unseen data.

6. Discussion

The application of PCA led to a significant reduction in feature space, lowering computational load while preserving important patterns. The classification performance, while not perfect, validated that compressed data retained enough distinguishing features for effective learning. Visualizations supported both the interpretability and reliability of the PCA approach.

Potential improvements could include:

- Exploring alternative dimensionality reduction methods (e.g., t-SNE, UMAP).
- Using more complex classifiers such as Random Forest or SVM.
- Increasing training data or fine-tuning PCA component count for better results.

7. Conclusion

This project demonstrated the practical value of PCA in compressing image data for classification tasks. Key takeaways include:

- Efficient dimensionality reduction from 64 to 30 features.
- High visual fidelity in reconstructed images.
- Successful classification using Logistic Regression on compressed data.
  This approach can be applied to various image processing scenarios where reducing data size without significant quality loss is essential.

References

- Scikit-learn Documentation: https://scikit-learn.org
- NumPy Documentation: https://numpy.org
- Matplotlib Documentation: https://matplotlib.org
- Seaborn Documentation: https://seaborn.pydata.org