PCA Assignments

MTech in Applied AI
Deep Learning Techniques

by

Sanjeev Kumar Pandey*
Faculty: Dr Anamika Gupta

Submitted June 4, 2025

* Student ID: 31050, Enrolment No:MT23AAI001, Email: pandey.sanjeev@yahoo.com

Table of Contents

1 PCA Programming Problems

1.1 Problem 1: Explained Variance Analysis using PCA

Objective

The objective of this task is to analyze how much variance is retained when using different numbers of principal components via Principal Component Analysis (PCA).

Instructions

- Load a dataset, such as the `Wine` or `Breast Cancer` dataset from `sklearn.datasets`.
- Standardize the dataset using `StandardScaler`.
- Use `PCA().fit()` to compute the explained variance ratios.
- Plot a cumulative explained variance vs. number of components curve.
- Identify the minimum number of components required to retain at least 95% of the variance.

1.1.1 Solution: Python code

Here is variance_analysis functions to to take input of breast cancer dataset and wine dataset for Variance Analysis using PCA :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_wine
from sklearn.datasets import load_breast_cancer

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

def variance_analysis(data_set, plot_name):
    print(f'dataset shape {data_set.data.shape}')
    X = data_set.data
    y = data_set.target

    # Standardize the features
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Apply PCA
    pca = PCA()
    pca.fit(X_scaled)

    # Calculate explained variance ratios and cumulative variance
    explained_variance_ratio = pca.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance_ratio)

    # Find the minimum number of components to retain 95% variance
    n_components_95 = np.argmax(cumulative_variance >= 0.95) + 1

    # Plot cumulative variance vs. number of components
    plt.figure(figsize=(10, 6))
    plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='-', color=
    plt.axhline(y=0.95, color='r', linestyle='--', label='95% Variance Threshold')
    plt.axvline(x=n_components_95, color='g', linestyle='--', label=f'Min Components: {n_components_95}')
    plt.title('Cumulative Explained Variance vs. Number of Components')
    plt.xlabel('Number of Components')
    plt.ylabel('Cumulative Explained Variance')
    plt.grid(True)
```

```
    plt.legend()
    plt.savefig(plot_name)

    # Print results
    print(f"Explained Variance Ratios: {explained_variance_ratio}")
    print(f"Cumulative Variance: {cumulative_variance}")
    print(f"Minimum number of components to retain 95% variance: {n_components_95}")

# Load and preprocess the Wine dataset
print(f'Analysing Wine Dataset')
data_w = load_wine()
variance_analysis(data_w, "pca_win.png")
print()
print(f'============================================================')
print(f'Analysing Breast Cancer Dataset')
data_b = load_breast_cancer()
variance_analysis(data_b, "pca_breast_cancer.png")
```

Listing 1: Code

1.1.2 Output Problem 1

```
C:\Users\urssa\AppData\Local\Programs\Python\Python311\python.exe C:\Sanjeev\VNIT_CLAS
Analysing Wine Dataset
dataset shape (178, 13)
Explained Variance Ratios: [0.36198848 0.1920749  0.11123631 0.0706903  0.06563294 0.0
 0.04238679 0.02680749 0.02222153 0.01930019 0.01736836 0.01298233
 0.00795215]
Cumulative Variance: [0.36198848 0.55406338 0.66529969 0.73598999 0.80162293 0.8509811
 0.89336795 0.92017544 0.94239698 0.96169717 0.97906553 0.99204785
 1.        ]
Minimum number of components to retain 95% variance: 10

============================================================
Analysing Breast Cancer Dataset
dataset shape (569, 30)
Explained Variance Ratios: [4.42720256e-01 1.89711820e-01 9.39316326e-02 6.60213492e-0
 5.49576849e-02 4.02452204e-02 2.25073371e-02 1.58872380e-02
 1.38964937e-02 1.16897819e-02 9.79718988e-03 8.70537901e-03
 8.04524987e-03 5.23365745e-03 3.13783217e-03 2.66209337e-03
 1.97996793e-03 1.75395945e-03 1.64925306e-03 1.03864675e-03
 9.99096464e-04 9.14646751e-04 8.11361259e-04 6.01833567e-04
 5.16042379e-04 2.72587995e-04 2.30015463e-04 5.29779290e-05
 2.49601032e-05 4.43482743e-06]
Cumulative Variance: [0.44272026 0.63243208 0.72636371 0.79238506 0.84734274 0.8875879
 0.9100953  0.92598254 0.93987903 0.95156881 0.961366   0.97007138
 0.97811663 0.98335029 0.98648812 0.98915022 0.99113018 0.99288414
 0.9945334  0.99557204 0.99657114 0.99748579 0.99829715 0.99889898
 0.99941502 0.99968761 0.99991763 0.99997061 0.99999557 1.        ]
Minimum number of components to retain 95% variance: 10

Process finished with exit code 0
```

Listing 2: Code

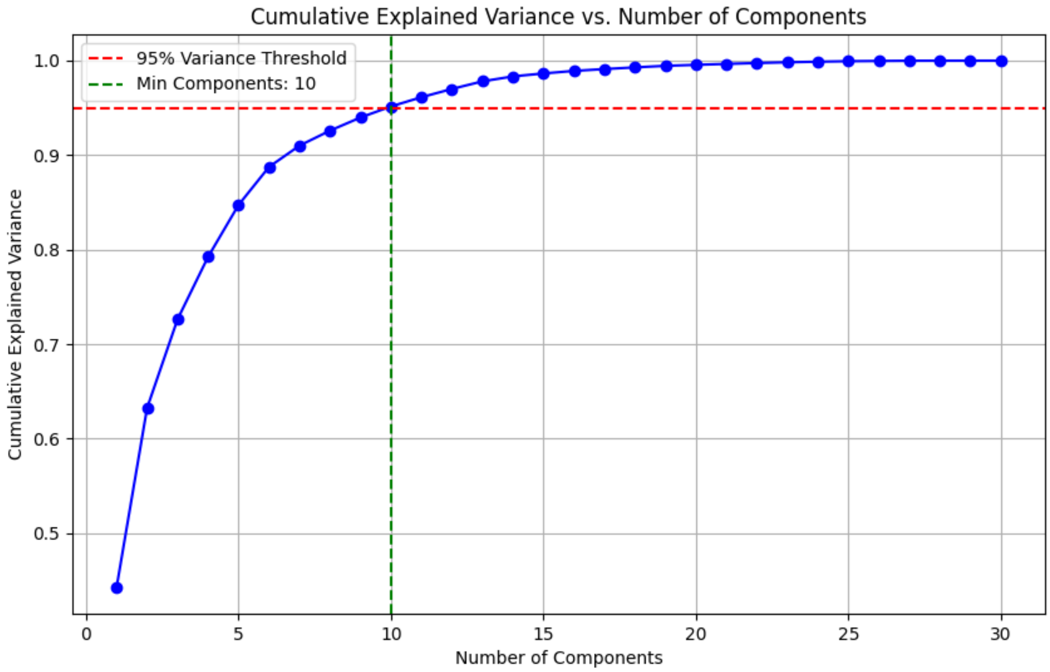

Figure 1: Output: Problem 1

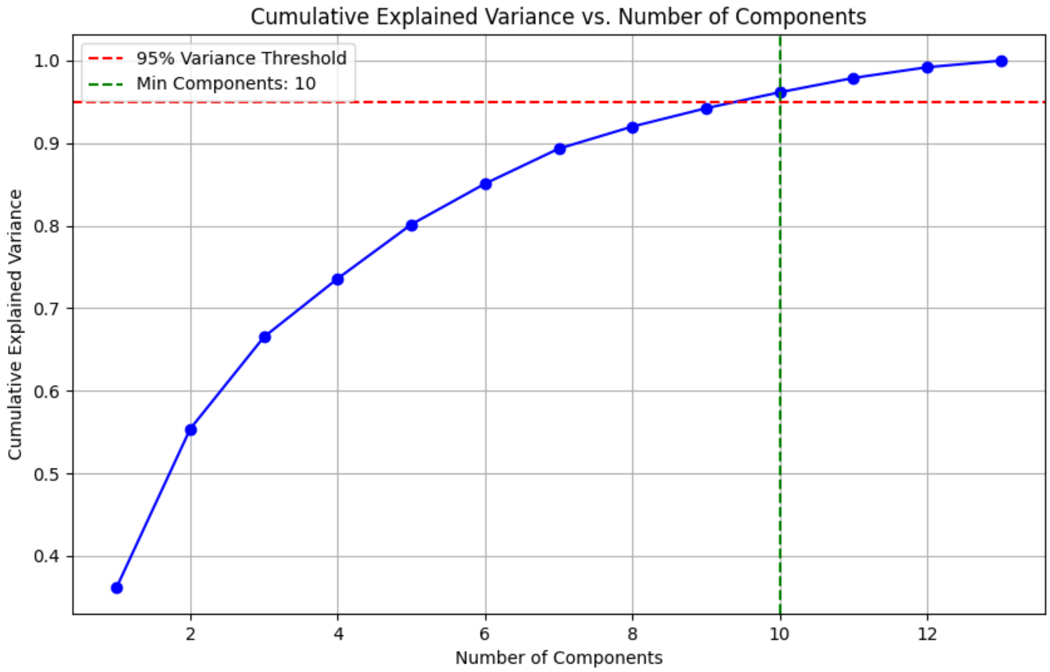

Figure 2: Output: Problem 1

2 Problem 2: Image Compression with PCA

Objective

Apply Principal Component Analysis (PCA) to compress and reconstruct grayscale images. For example, use the `camera()` image from `skimage.data` or a sample from the MNIST dataset.

Steps

1. Convert the image into a 2D array (if not already in that form).
2. Apply PCA to reduce dimensionality (number of components).
3. Reconstruct the image using the inverse PCA transform.
4. Visualize and compare the original image with its reconstruction.

Bonus

Try compressing the image using different numbers of components (e.g., 5, 20, 50) and observe how reconstruction quality changes.

2.0.1 Solution: Python code

```
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from skimage import data
import numpy as np
```

```

# Load and normalize the grayscale image
image = data.camera()  # shape: (512, 512), grayscale
X = image / 255.0      # Normalize pixel values to [0, 1]

# Function to compress and reconstruct the image using PCA
def pca_image_compression(X, n_components):
    pca = PCA(n_components=n_components)
    X_transformed = pca.fit_transform(X)
    X_reconstructed = pca.inverse_transform(X_transformed)
    return X_reconstructed

# List of PCA component counts to test
components_list = [5, 20, 50]

# Total plots = 1 original + len(components_list)
plt.figure(figsize=(15, 4))

# Plot original image
plt.subplot(1, len(components_list) + 1, 1)
plt.imshow(X, cmap='gray')
plt.title("Original Image")
plt.axis('off')

# Plot reconstructed images
for i, n in enumerate(components_list):
    reconstructed = pca_image_compression(X, n)
    plt.subplot(1, len(components_list) + 1, i + 2)
    plt.imshow(reconstructed, cmap='gray')
    plt.title(f'{n} Components')
    plt.axis('off')

plt.suptitle("Original vs Reconstructed Images using PCA")
plt.tight_layout()
plt.savefig('problem_2.png')
```

Listing 3: Code

2.0.2 Output Problem 2

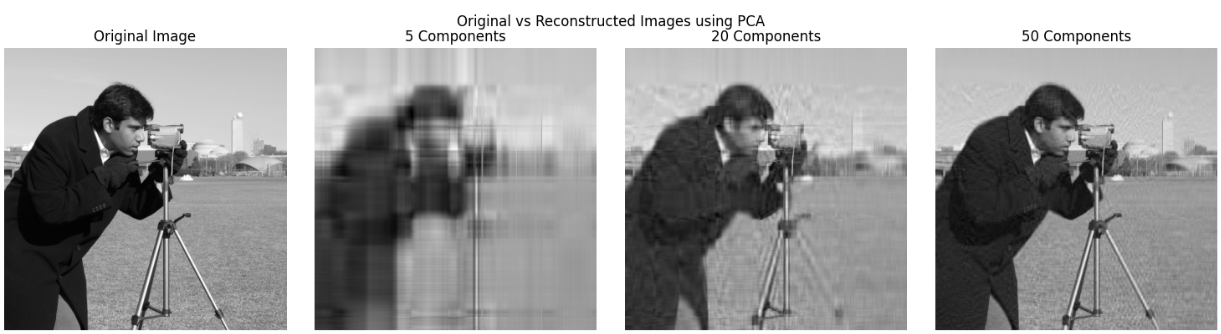

Figure 3: Output: Problem 2

3 End of Document