Analiza Głównych Składowych – PCA

Liliana Sirko, Miłosz Zieliński

February 25, 2025

1 Czym jest PCA? Zastosowania.

Analiza głównych składowych (PCA) to metoda redukcji wymiarowości danych, która pozwala na zachowanie najważniejszych informacji. Wynikiem PCA są główne składowe, czyli wektory własne macierzy kowariancji uzyskanej z macierzy danych. Wskazują one kierunki, w których dane najbardziej różnią się od wartości średniej (gdzie wariancja jest największa). W przypadku zastosowania PCA na obrazie, główne składowe mogą m.in. wskazywać krawędzie obiektów na nim przedstawionych. Ponieważ wynikowe wektory są ortogonalne, nie ma żadnych kowariancji pomiędzy głównymi składowymi. Dzięki temu uzyskujemy strukturę danych, która jest łatwiejsza do zrozumienia i wizualizacji oraz lepiej przystosowana do dalszej analizy.

PCA ma szerokie zastosowania w analizie danych, np.:
- Wizualizacja danych o więcej niż trzech wymiarach: dane wielowymiarowe trudno jest przedstawić na grafie 2D lub 3D. PCA umożliwia przedstawienie danych w dwóch wymiarach tak, aby większość informacji z pierwotnych danych została zachowana. Grupy i wariancje istniejące w danych pozostają widoczne na płaszczyźnie.
- Analiza tekstu: gdy dane tekstowe są zamieniane na liczbowe, mają bardzo dużo wymiarów. Na przykład, dla algorytmu bag-of-words, każde unikalne słowo w tekście ma swój własny wymiar. Aby efektywnie wykorzystać dane tekstowe w przetwarzaniu języka naturalnego (NLP), np. do wykrywania spamu w emailach, można użyć PCA.
- Rozpoznawanie twarzy (eigenfaces): poprzez wykonanie PCA na dużym zbiorze zdjęć twarzy można wygenerować tzw. eigenfaces. Są to składowe główne, które poprzez zmianę ich wag można połączyć w twarze spoza pierwotnego zbioru. Dzięki temu dane o twarzach można przechowywać z dużą dokładnością bez przechowywania samych zdjęć. Do odtworzenia kilkuset twarzy może wystarczyć zaledwie 30 eigenfaces.
- Kompresja obrazów: PCA pozwala zaoszczędzić na liczbie przechowywanych pikseli, zachowując przy tym najważniejsze elementy obrazu.

2 Działanie algorytmu

Dla macierzy $X = [x_{ij}] \in M_m^n(R)$
1. Wyliczamy średnią wartości w rzędach: $\bar{x}_j = \frac{1}{n} \sum_{i=1} x_{ij}$
2. Wyliczamy macierzy odchyleń B: od od każdej wartości x_{ij} macierzy X odejmujemy średnią $\bar{x}_j$
3. Wyliczamy macierz kowariancji: $C = \frac{1}{n-1} B^* B$
4. Następnie postępujemy zgodnie z algorytmem SVD (dla macierzy C).

3 Działanie algorytmu na przykładzie

Postanowiliśmy pokazać działanie algorytmu PCV poprzez znaczące zmniejszenie ilości pixeli potrzebnych do zapisania obrazu, przy jednoczesnej niewielkiej stracie na jakości obrazu.
W poniższym przykładzie wykorzystaliśmy znane zdjęcie fotoreportera Steva McCurrego "Afghan Girl".
Wszytskie kroki procesu opisane są przy odpowiednich fragmentach kodu. Warto jednak zwrócić uwagę na kilka rzeczy.
Obraz jest kolorowy, daltego wszystkie działania przeprowadzaliśmy trzy razy, dla macierzy poszczególnych kolorów, po czym złączyliśmy je na końcu.
Początkowy obraz miał 600 wymiarów, a końcowy, po wybraniu 50 największych wartości własnych z każdej macierzy, tylko 50. Nastąpiła więc bardzo duża redukcja, przy czym jakość obrazu nie zmniejszyła się bardzo mocno. Możemy jednak zauważyć anomalie, kilka plam kolorów.
Ciekawe jest, że przy zapisie obrazu w formacie .jpg jego rozmiar w pamięci jest bardzo podobnej wielkości, co obrazu początkowego. Wynika to z procesu zapisu do formatu jpg. Jendak gdy obraz zapisywaliśmy obraz w formacie png, to rzeczywiście zajmował on mniej miejsca od początkowego obrazu w formacie png (o około 20%).

[1]:
```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

# Załadowanie obrazu
image = Image.open('AfghanGirl.jpg')

# Konwertowanie obrazu do formatu NumPy
image_np = np.array(image)

# Rozdzielenie kanałów
matrix_red = image_np[:, :, 0]
matrix_green = image_np[:, :, 1]
matrix_blue = image_np[:, :, 2]

# Ustawienia wyświetlania
fig, axes = plt.subplots(1, 4, figsize=(20, 5))

# Oryginalny obraz
axes[0].imshow(image)
axes[0].set_title("Oryginalny obraz")
axes[0].axis('off')

# Kanał czerwony
axes[1].imshow(matrix_red)
axes[1].set_title("Kanał czerwony")
axes[1].axis('off')

# Kanał zielony
#cmap='Greens' - after green_chanel
axes[2].imshow(matrix_green)
axes[2].set_title("Kanał zielony")
axes[2].axis('off')

# Kanał niebieski
axes[3].imshow(matrix_blue)
axes[3].set_title("Kanał niebieski")
axes[3].axis('off')

# Pokaż obrazy
plt.show()
```

[2]:
```python
# Funkcja do wyznaczania średnich dla wierszy
def calculate_row_means(matrix):
    row_means = np.mean(matrix, axis=1)
    return row_means

# Funkcja do wyliczania macierzy odchyleń
def calculate_deviations(matrix, row_means):
    deviations = matrix - row_means[:, np.newaxis]
    return deviations

# Funkcja do wyznaczania macierzy kowariancji
def calculate_covariance_matrix(deviations):
    covariance_matrix = np.dot(deviations, deviations.T) / deviations.shape[1]
    return covariance_matrix


# Wyznaczanie średnich dla wierszy dla każdej z macierzy
row_means_red = calculate_row_means(matrix_red)
row_means_green = calculate_row_means(matrix_green)
row_means_blue = calculate_row_means(matrix_blue)

# Wyliczanie macierzy odchyleń dla każdej z macierzy
deviations_red = calculate_deviations(matrix_red, row_means_red)
deviations_green = calculate_deviations(matrix_green, row_means_green)
deviations_blue = calculate_deviations(matrix_blue, row_means_blue)

# Wyznaczanie macierzy kowariancji dla każdej z macierzy odchyleń
covariance_matrix_red = calculate_covariance_matrix(deviations_red)
covariance_matrix_green = calculate_covariance_matrix(deviations_green)
covariance_matrix_blue = calculate_covariance_matrix(deviations_blue)
```

[25]:
```python
# Funkcja do policzenia wartości własnych i wektorów własnych
def calculate_eigenvalues_and_vectors(covariance_matrix):
    eigenvalues, eigenvectors = np.linalg.eigh(covariance_matrix)
    return eigenvalues, eigenvectors
```

```
# Funkcja do wybrania największych wartości i wektorów własnych
def select_top_eigenvalues(eigenvalues, eigenvectors, top_n=50):
    sorted_indices = np.argsort(eigenvalues)[::-1]
    top_eigenvalues = eigenvalues[sorted_indices][:top_n]
    top_eigenvectors = eigenvectors[:, sorted_indices][:, :top_n]
    return top_eigenvalues, top_eigenvectors

# Liczenie wartości i wektoró własnych
eigenvalues_red, eigenvectors_red =␣
↪calculate_eigenvalues_and_vectors(covariance_matrix_red)
eigenvalues_green, eigenvectors_green =␣
↪calculate_eigenvalues_and_vectors(covariance_matrix_green)
eigenvalues_blue, eigenvectors_blue =␣
↪calculate_eigenvalues_and_vectors(covariance_matrix_blue)

# Wybranie największych 50 wartości i wektorów własnych
top_eigenvalues_red, top_eigenvectors_red =␣
↪select_top_eigenvalues(eigenvalues_red, eigenvectors_red)
top_eigenvalues_green, top_eigenvectors_green =␣
↪select_top_eigenvalues(eigenvalues_green, eigenvectors_green)
top_eigenvalues_blue, top_eigenvectors_blue =␣
↪select_top_eigenvalues(eigenvalues_blue, eigenvectors_blue)
```

[36]:
```
# Funkcja do rzutowania na wektory własne
def project_onto_eigenvectors(matrix, eigenvectors):
    projected_data = np.dot(eigenvectors.T, matrix)
    return projected_data

# Rzutowanie na wybrane wektory własne
projected_data_red = project_onto_eigenvectors(deviations_red,␣
↪top_eigenvectors_red)
projected_data_green = project_onto_eigenvectors(deviations_green,␣
↪top_eigenvectors_green)
projected_data_blue = project_onto_eigenvectors(deviations_blue,␣
↪top_eigenvectors_blue)

# Odrzutowanie do oryginalnej przestrzeni
reconstructed_data_red = np.dot(top_eigenvectors_red, projected_data_red)
reconstructed_data_green = np.dot(top_eigenvectors_green, projected_data_green)
reconstructed_data_blue = np.dot(top_eigenvectors_blue, projected_data_blue)

# Dodanie średnich z powrotem
reconstructed_data_red += row_means_red[:, np.newaxis]
reconstructed_data_green += row_means_green[:, np.newaxis]
reconstructed_data_blue += row_means_blue[:, np.newaxis]
```

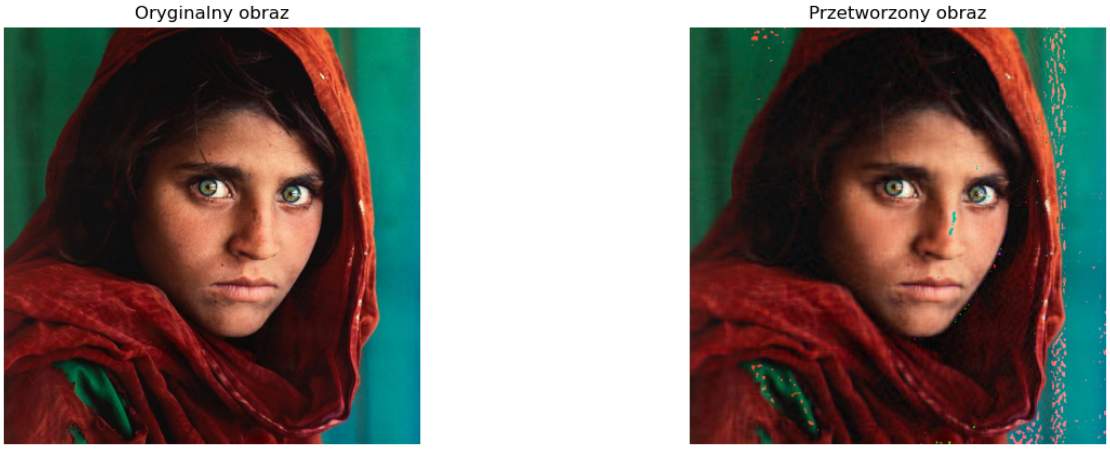
```
# Składanie przetworzonych kanałów z powrotem do obrazu
projected_image = np.zeros_like(image_np)
projected_image[:, :, 0] = reconstructed_data_red
projected_image[:, :, 1] = reconstructed_data_green
projected_image[:, :, 2] = reconstructed_data_blue

# Wyświetlenie oryginalnego i przetworzonego obrazu
fig, axes = plt.subplots(1, 2, figsize=(15, 5))

# Oryginalny obraz
axes[0].imshow(image)
axes[0].set_title("Oryginalny obraz")
axes[0].axis('off')

# Przetworzony obraz
axes[1].imshow(projected_image.astype(np.uint8))
axes[1].set_title("Przetworzony obraz")
axes[1].axis('off')

plt.show()
```



[38]:
```
# Zapis przetworzonego obrazu
processed_image = Image.fromarray(projected_image.astype(np.uint8))
processed_image.save('AfghanGirl_processed.jpg')
```