Proyecto Final Reconocimiento de Rostros - Eigenfaces

Julio Pérez
Estudiante, Facultad de Ingeniería
Ciudad Universitaria, Ciudad de México, México

Mauricio Bautista
Estudiante, Facultad de Ingeniería
Ciudad Universitaria Ciudad de México, México

***Resumen*—En el documento se explicará cómo se estructuro el algoritmo de análisis de componentes principales con el fin de hacer una reconstrucción facial.**

Palabras clave—PCA, Vectores, Covarianza

I. INTRODUCCIÓN

El Análisis de Componentes Principales (PCA, por sus siglas en inglés) es una técnica de reducción de dimensionalidad ampliamente utilizada en análisis de datos y machine learning. Su principal objetivo es transformar un conjunto de datos con muchas variables correlacionadas en un conjunto de variables no correlacionadas denominadas componentes principales, que capturan la mayor parte de la variabilidad presente en los datos originales. Esta técnica es especialmente útil cuando se trabaja con datos de alta dimensionalidad, ya que permite simplificar el modelo sin perder información significativa..

II. LIBRERIAS

El código emplea varias bibliotecas de Python para diferentes tareas: os para interactuar con el sistema operativo, especialmente para listar archivos y construir rutas; numpy (np) para operaciones matemáticas y de manipulación de matrices; PIL (Pillow) para abrir, manipular y convertir imágenes a escala de grises; cv2 (OpenCV) para procesamiento de imágenes, incluyendo la detección de rostros utilizando clasificadores Haar Cascade; sklearn (scikit-learn) para aplicar el Análisis de Componentes Principales (PCA) y estandarizar los datos con StandardScaler, además de calcular el error cuadrático medio con mean_squared_error; y matplotlib.pyplot para crear visualizaciones de los datos y los resultados del PCA.

```python
import os
import numpy as np
from PIL import Image
import cv2  # Asegúrate de tener OpenCV instalado
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

Fig.1 Bibliotecas usadas

III. CARGA Y CLACIFICACION DE IMAGENES

La función detect_face(image) tiene como propósito detectar y recortar el rostro de una imagen dada. Para lograr esto, convierte la imagen a una matriz de numpy en escala de grises y utiliza el clasificador Haar Cascade de OpenCV para detectar rostros en la imagen. El clasificador escanea la imagen y devuelve las coordenadas de los rostros detectados. Si se encuentra al menos un rostro, la función recorta la región correspondiente al rostro y la devuelve. Este recorte permite enfocar el análisis en la parte relevante de la imagen, que en este caso son los rostros.

La función load_images_from_folder(folder) se encarga de cargar y procesar todas las imágenes contenidas en un directorio especificado. Itera sobre todos los archivos en el directorio, abre cada imagen y la convierte a escala de grises. Luego, utiliza la función detect_face para detectar y recortar los rostros en cada imagen. Si se detecta un rostro, la imagen se redimensiona a un tamaño uniforme de 100x100 píxeles y se aplana para formar un vector de características. Este proceso de carga y preprocesamiento convierte las imágenes en datos numéricos adecuados para el análisis posterior con PCA.

La función load_specific_images(image_paths) es similar a load_images_from_folder, pero en lugar de cargar todas las imágenes de un directorio, carga y procesa un conjunto específico de imágenes cuyos caminos se proporcionan como entrada. Esta función abre cada imagen, la convierte a escala de grises, y utiliza detect_face para detectar y recortar el rostro. Si se detecta un rostro, la imagen se redimensiona y aplana. Esta función es para preparar las imágenes a ser reconstruidas.

```python
# Cargar las imágenes de entrenamiento
faces_train = load_images_from_folder(dataset_path)
print("Imágenes cargadas\n")
n_samples, n_features = faces_train.shape

# Cargar las imágenes específicas a reconstruir
faces_test = load_specific_images(image_paths)
print("Imagenes para reconstruir cargadas\n")
```

Fig.2 Funciones en el flujo principal

```python
def detect_face(image):
    print("Recortando rostros")
    gray = np.array(image)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
    if len(faces) == 0:
        return None
    (x, y, w, h) = faces[0]
    face = gray[y:y+h, x:x+w]
    return face

def load_images_from_folder(folder):
    print("\nCargando imagenes desde el folder dataset\n")
    images = []
    for filename in os.listdir(folder):
        if filename.endswith(".jpg") or filename.endswith(".bmp"):
            img_path = os.path.join(folder, filename)
            img = Image.open(img_path).convert('L')  # Convertir a escala de grises
            face = detect_face(img)
            if face is not None:
                face = Image.fromarray(face).resize((100, 100))  # Redimensionar la imagen del
rostro
                img_array = np.array(face).flatten()  # Aplanar la imagen
                images.append(img_array)
    return np.array(images)

def load_specific_images(image_paths):
    print("Cargando imagenes que serán reconstruidas\n")
    images = []
    for img_path in image_paths:
        img = Image.open(img_path).convert('L')  # Convertir a escala de grises
        face = detect_face(img)
        if face is not None:
            face = Image.fromarray(face).resize((100, 100))  # Redimensionar la imagen del rostro
            img_array = np.array(face).flatten()  # Aplanar la imagen
            images.append(img_array)
    return np.array(images)
```

Fig.3 funciones para cargar y reconocer imágenes

IV. Estandarización de los Datos de Entrenamiento

La estandarización de los datos de entrenamiento se realiza utilizando StandardScaler de sklearn, que ajusta el escalador a los datos calculando la media y la desviación estándar de cada característica y luego transforma los datos para que cada característica tenga media cero y varianza unitaria. Esto se hace mediante el método fit_transform, que primero ajusta el escalador y luego aplica la transformación. Estandarizar los datos es crucial para PCA, ya que esta técnica es sensible a las escalas de las características y la estandarización asegura que todas las características contribuyan de manera equitativa al análisis.

```python
# Estandarizar los datos de entrenamiento
scaler = StandardScaler()
faces_train_std = scaler.fit_transform(faces_train)
```

Fig.4 Estandarización de los datos de entrenamiento.

V. Cálculo de la Matriz de Covarianza

El cálculo de la matriz de covarianza es un paso esencial en PCA, ya que esta matriz describe la varianza y las covarianzas entre pares de características en los datos estandarizados. La matriz se calcula utilizando np.cov(faces_train_std.T), donde la transposición de los datos es necesaria para que cada fila represente una variable y cada columna una observación. La matriz de covarianza resultante tiene un tamaño igual al número de características en cada imagen, lo que permite identificar las direcciones de máxima varianza en los datos.

```python
# Calcular la matriz de covarianza
print("Generando matriz de covarianza\n")
cov_matrix = np.cov(faces_train_std.T)
print(len(cov_matrix))
```

Fig.5 matiz de covarianza

VI. Descomposición en Valores y Vectores Propios

La descomposición en valores y vectores propios se realiza utilizando np.linalg.eigh en la matriz de covarianza. Esta función devuelve los valores propios (eigenvalues), que indican la cantidad de varianza explicada por cada componente, y los vectores propios (eigenvectors), que son las direcciones principales de variación en los datos. Los vectores propios obtenidos son ortogonales entre sí y representan los componentes principales que se utilizarán para proyectar los datos en un espacio de menor dimensionalidad.

```python
# Realizar la descomposición en valores propios
print("Descomposicion en valores propios\n")
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
```

Fig.6 Obtención de los valores y vectores propios

VII. Ordenar los Valores y Vectores Propios

Para ordenar los valores y vectores propios en orden descendente de importancia, se utilizan los índices ordenados de los valores propios obtenidos mediante np.argsort(eigenvalues). Estos índices se utilizan para reordenar tanto los valores propios como los vectores propios, asegurando que los primeros componentes seleccionados sean los que explican la mayor cantidad de varianza en los datos. Este reordenamiento es crucial para seleccionar los primeros k componentes principales que capturan la mayor variabilidad, permitiendo una reducción efectiva de la dimensionalidad mientras se retiene la mayor cantidad de información posible.

```python
# Ordenar los valores y vectores propios
print("Ordenar los valores y vectores propios\n")
sorted_index = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_index]
eigenvectors = eigenvectors[:, sorted_index]
```

Fig.6 Ordenar los valores y vectores propios para el modelo

VIII. Selección de los Primeros k Componentes Principales

En este paso, se seleccionan los primeros k componentes principales. Aquí, k se establece en 50, lo que significa que se seleccionarán los 50 vectores propios que corresponden a los 50 valores propios más grandes. Esta selección se realiza tomando las primeras 50 columnas de la matriz de vectores propios (eigenvectors[: , :k]). Los componentes principales seleccionados son las direcciones en las cuales los datos tienen la mayor varianza, y reducir el número de componentes a 50

ayuda a simplificar el modelo mientras se retiene la mayor cantidad de información posible.

Fig.7 Componentes elegidos

IX. Proyección de las Imágenes Específicas en el Nuevo Espacio

Primero, las imágenes de prueba (faces_test) se estandarizan utilizando el mismo escalador (scaler) que se utilizó para las imágenes de entrenamiento. Esto asegura que las imágenes de prueba tengan la misma escala que las de entrenamiento. Luego, las imágenes estandarizadas se proyectan en el espacio de componentes principales utilizando una multiplicación de matrices (@) con los vectores propios seleccionados. Esta proyección transforma las imágenes de prueba en el nuevo espacio de menor dimensionalidad definido por los componentes principales seleccionados.

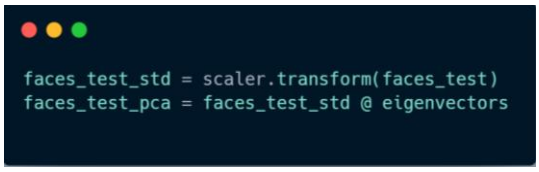

Fig.8 Estandarización de los datos de entrenamiento y creando su espacio

X. Reconstrucción de las Imágenes a partir de los Componentes Principales

Para reconstruir las imágenes proyectadas en el espacio original, se realiza una multiplicación de matrices inversa (@ eigenvectors.T) que proyecta los datos de vuelta al espacio original de mayor dimensionalidad. Sin embargo, estas imágenes reconstruidas aún están en la escala estandarizada, por lo que se utiliza scaler.inverse_transform para revertir la estandarización y obtener las imágenes en su escala original. Este proceso de reconstrucción permite evaluar cómo de bien los componentes principales seleccionados pueden aproximar las imágenes originales y es crucial para entender la efectividad de la reducción de dimensionalidad.

Fig.9 Cálculos inversos para regresar al espacio original.

XI. Calcular el MSE para cada Imagen (Implementación Propia)

En esta parte del código, se calcula el error medio cuadrático (MSE) entre las imágenes originales de prueba y las imágenes reconstruidas. El MSE es una medida que cuantifica la diferencia promedio al cuadrado entre los valores predichos (reconstruidos) y los valores reales (originales). Aquí, se utiliza una lista por comprensión para iterar sobre todas las imágenes de prueba, calculando el MSE individual para cada una usando la función mean_squared_error de sklearn.metrics. Luego, se calcula el MSE promedio de todas las imágenes para obtener una medida general de la calidad de la reconstrucción.

Fig.10 MSE para evaluar la fiabilidad

XII. Uso de PCA de scikit-learn para Comparación

En esta sección, se utiliza la implementación de PCA de la biblioteca sklearn para realizar la misma tarea de reducción de dimensionalidad y reconstrucción, permitiendo una comparación directa con la implementación propia. Primero, se instancia un objeto PCA con n_components=k para especificar el número de componentes principales. Luego, se ajusta el modelo a los datos estandarizados de entrenamiento (fit), y se transforman las imágenes de prueba estandarizadas al espacio de componentes principales (transform). Las imágenes transformadas se reconstruyen en el espacio original utilizando inverse_transform. Finalmente, las imágenes reconstruidas se desescalan a su escala original con scaler.inverse_transform.

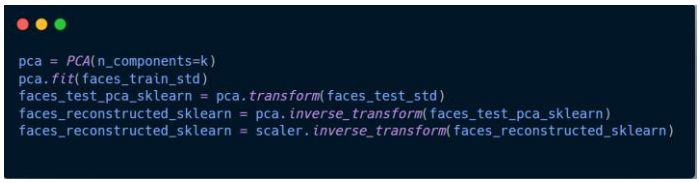

Fig.11 PCA de SCIKIT-LEARN para comparar

XIII. Calcular el MSE para cada Imagen (sklearn)

Se calcula el MSE para las imágenes reconstruidas usando la implementación de PCA de sklearn de la misma manera que se hizo con la implementación propia. Se utiliza una lista por comprensión para calcular el MSE individual para cada imagen de prueba y luego se calcula el MSE promedio para todas las imágenes. Este valor también se imprime para comparar directamente la calidad de la reconstrucción entre la implementación propia y la de sklearn. Comparar estos valores de MSE promedio proporciona una evaluación clara de la precisión y eficacia de ambas implementaciones de PCA.

Fig.12 MSE para evaluar la fiabilidad de PCA de SCIKIT-LEARN

XIV. Visualizacion de imágenes

Al final el código visualiza las imágenes originales de prueba junto con sus versiones reconstruidas utilizando tanto la implementación propia de PCA como la de sklearn. Se crean figuras con subplots donde se muestran las imágenes originales en la primera columna y sus correspondientes reconstrucciones en la segunda columna, permitiendo comparar visualmente la calidad de la reconstrucción. Esto se realiza para las dos primeras imágenes de prueba, facilitando una comparación

directa entre las imágenes originales y las reconstruidas por ambos métodos. Cada imagen se transforma en una matriz de 100x100 píxeles y se muestra en escala de grises, proporcionando una visión clara de las diferencias y similitudes en la reconstrucción.

```python
# Visualizar las imágenes originales y reconstruidas
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

for i in range(2):
    ax[i, 0].imshow(faces_test[i].reshape((100, 100)), cmap='gray')
    ax[i, 0].set_title('Original')
    ax[i, 0].axis('off')

    ax[i, 1].imshow(faces_reconstructed[i].reshape((100, 100)), cmap='gray')
    ax[i, 1].set_title('Reconstruida')
    ax[i, 1].axis('off')

plt.show()

# Visualizar las imágenes reconstruidas con scikit-learn PCA
fig, ax = plt.subplots(2, 2, figsize=(10, 10))

for i in range(2):
    ax[i, 0].imshow(faces_test[i].reshape((100, 100)), cmap='gray')
    ax[i, 0].set_title('Original')
    ax[i, 0].axis('off')

    ax[i, 1].imshow(faces_reconstructed_sklearn[i].reshape((100, 100)), cmap='gray')
    ax[i, 1].set_title('Reconstruida (sklearn)')
    ax[i, 1].axis('off')

plt.show()
```

Fig.13 Muestra de comparación de los PCA

XV. Visualización de Gráficas de Varianza y Componentes Principales

Se grafica la proporción acumulada de varianza explicada por los componentes principales, mostrando cómo la varianza acumulada aumenta con el número de componentes, lo que ayuda a decidir cuántos componentes son necesarios para explicar una cantidad significativa de la varianza en los datos. Además, se visualizan los primeros componentes principales como imágenes en escala de grises, mostrando las "caras" promedio que estos componentes representan. Finalmente, se comparan las matrices de covarianza de las imágenes originales y reconstruidas mediante mapas de calor. Las matrices de covarianza de las imágenes reconstruidas utilizando tanto la implementación propia como la de sklearn se muestran lado a lado con la matriz de covarianza original, permitiendo una comparación visual de cómo las reconstrucciones preservan las relaciones de varianza entre los píxeles.

```python
explained_variance_ratio = np.cumsum(eigenvalues / np.sum(eigenvalues))

plt.figure(figsize=(8, 5))
plt.plot(explained_variance_ratio, marker='o')
plt.xlabel('Número de Componentes Principales')
plt.ylabel('Proporción de Varianza Explicada')
plt.title('Proporción de Varianza Explicada por los Componentes Principales')
plt.grid(True)
plt.show()

# Visualización de los primeros componentes principales
num_components_to_show = 10
fig, axes = plt.subplots(1, num_components_to_show, figsize=(20, 4))
for i in range(num_components_to_show):
    ax = axes[i]
    ax.imshow(eigenvectors[:, i].reshape(100, 100), cmap='gray')
    ax.set_title(f'Componente {i+1}')
    ax.axis('off')
plt.suptitle('Primeros Componentes Principales')
plt.show()

# Comparación de las matrices de covarianza
cov_matrix_test = np.cov(faces_test_std.T)
cov_matrix_reconstructed = np.cov(scaler.transform(faces_reconstructed).T)
cov_matrix_reconstructed_sklearn = np.cov(scaler.transform(faces_rconstructed_sklearn).T)

plt.figure(figsize=(15, 5))
plt.subplot(1, 3, 1)
plt.imshow(cov_matrix_test, cmap='hot', interpolation='nearest')
plt.title('Covarianza Original')
plt.colorbar()
plt.subplot(1, 3, 2)
plt.imshow(cov_matrix_reconstructed, cmap='hot', interpolation='nearest')
plt.title('Covarianza Reconstruida (Propia)')
plt.colorbar()
plt.subplot(1, 3, 3)
plt.imshow(cov_matrix_reconstructed_sklearn, cmap='hot', interpolation='nearest')
plt.title('Covarianza Reconstruida (sklearn)')
plt.colorbar()
plt.suptitle('Comparación de Matrices de Covarianza')
plt.show()
```

Fig.14 Graficas de los datos principales

XVI. Resultados

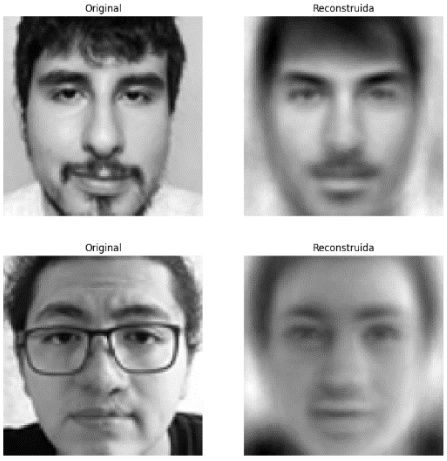

Fig.15 imágenes reconstruidas con nuestro PCA

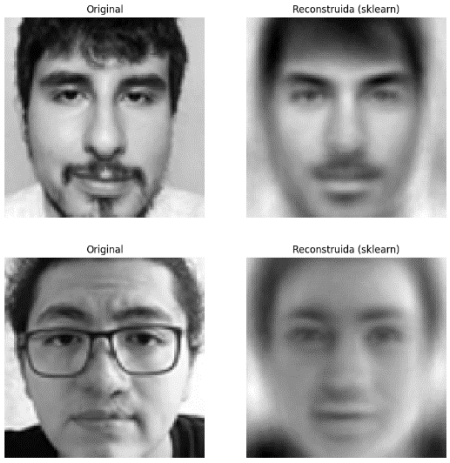

Fig.16 Imágenes reconstruidas con PCA de SCIKIT-LEARN

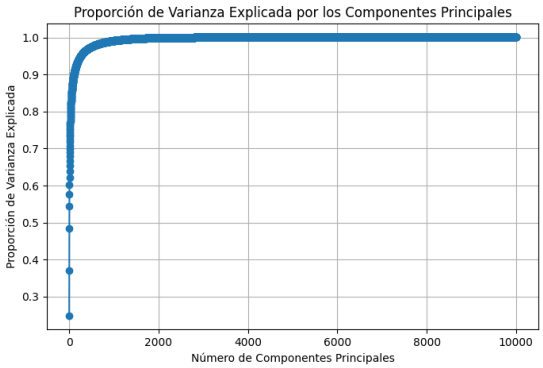

Fig.17 Grafica de la varianza contra los componentes

Fig.18 los primeros 10 componentes principales

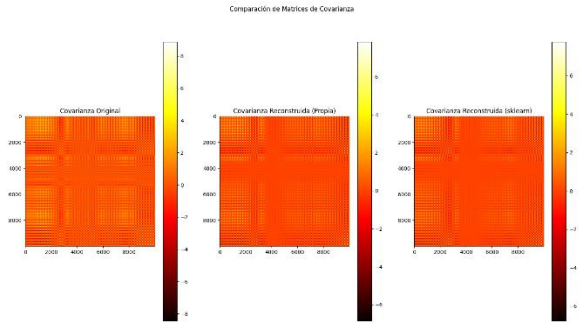

Fig.19 Comparación de las matrices

XVII. Comparacion de nuestro PCA contra AUTOENCODER

En la comparación, se observa que las reconstrucciones con PCA son más claras debido al uso del filtro Haar Cascade, que extrae solamente las regiones de interés de las imágenes. No obstante, autoencoder logran capturar características de manera más precisa y detallada. Esto sugiere que, si el objetivo es recolectar características particulares, autoencoder es superior. Sin embargo, para la tarea específica de reconstrucción y presentación de imágenes, PCA ha demostrado ofrecer mejores resultados en términos de claridad y fidelidad visual.

Fig.20 Reconstrucción de autoencoder

XVIII. Conclusión

En conclusión, hemos implementado y comparado dos métodos de análisis de componentes principales (PCA) para la reducción de dimensionalidad y reconstrucción de imágenes de rostros. Primero, utilizamos una implementación propia de PCA junto con la estandarización de datos y el cálculo de la matriz de covarianza, seguido de la descomposición en valores y vectores propios. Posteriormente, seleccionamos los componentes principales más significativos para proyectar y reconstruir las imágenes. Paralelamente, empleamos la implementación de PCA de scikit-learn para realizar una comparación directa. Evaluamos la calidad de las reconstrucciones calculando el error cuadrático medio (MSE) para ambos métodos.

Visualizamos las imágenes originales y reconstruidas, así como la proporción de varianza explicada por los componentes principales y las matrices de covarianza. A través de este proceso, hemos demostrado que PCA puede ser efectivo para la reconstrucción de imágenes, y al comparar con autoencoders, observamos que aunque los autoencoders capturan características más finas y detalladas, PCA ofrece una mayor claridad en las imágenes reconstruidas cuando se utilizan técnicas de preprocesamiento como el filtro Haar Cascade.

XIX. Referencias

[1] «Numpy.cov — NumPy v1.26 Manual». Disponible en: https://numpy.org/doc/stable/reference/generated/numpy.cov.html

[2] «Numpy.linalg.eigh — NumPy V1.26 Manual». [En línea]. Available: https://numpy.org/doc/stable/reference/generated/numpy.linalg.eigh.html.

[3] M. Badole, «Difference Between fit(), transform(), and fit_transform() Methods in Scikit-Learn», Analytics Vidhya, 13-sep-2023. [En línea]. Available: https://www.analyticsvidhya.com/blog/2021/04/difference-between-fit-transform-fit_transform-methods-in-scikit-learn-with-python-code/.

[4] «PCA», Scikit-learn. [En línea]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

[5] A. Rosebrock, «OpenCV haar cascades - PyImageSearch», PyImageSearch, 17-abr-2021. [En línea]. Available: https://pyimagesearch.com/2021/04/12/opencv-haar-cascades/.

[6] “Guia inicial de TensorFlow 2.0 para principiantes”, TensorFlow. [En línea]. Disponible en: https://www.tensorflow.org/tutorials/quickstart/beginner?hl=es-419. [Consultado: 19-may-2024].