PRÁCTICA 2 - BIOMETRÍA

PCA - EigenFaces

Cristian Villarroya Sánchez

01-05-2021

Índice

1. Descripción de la tarea 2
2. Carga de las imágenes 2
3. Cálculo de PCA 3
4. Variación de la dimensionalidad 4
5. Conclusiones 4
6. Bibliografía 4

1. Descripción de la tarea

Se disponen de 10 imágenes de 40 individuos diferentes, siendo las 5 primeras las que se usarán en entrenamiento y las 5 últimas en test.

El objetivo es implementar eigenfaces utilizando el vecino más cercano y obtener las curvas de error variando el valor de la dimensionalidad reducida d'.

2. Carga de las imágenes

Las imágenes vienen dadas en formato PGM. Este formato es un formato de archivo en escala de grises de mínimo denominador común. El dataset está organizado de la siguiente manera:

- \data
  - \Test
    - \s1
      - 6.pgm
      - …
    - \s2
    - …
    -
  - \Train
    - \s1
      - 1.pgm
      - …
    - \s2
    - …

Lo primero que se ha hecho es un método (read_images) mediante el cual, se accede a la carpeta raíz y se buscan todas las carpetas dentro de dicha carpeta raíz. Esto se hace para hacer al algoritmo invariante de tener 40 o 50 carpetas con imágenes de usuarios, facilitando de esta manera la adicción de datos en un futuro si fuese necesario.

Para cada una de las subcarpetas de la carpeta raíz, busca todos los ficheros, que serán los ficheros PGM de las imágenes de caras.

Para leer el fichero PGM, se ha hecho uso del método descrito en la página Intellipaat [1] (read_pgm). Este método devuelve un conjunto de 3 valores, el primero es un numpy array de 1xn con los datos, el segundo una tupla con la longitud y profundidad y el tercero el número de niveles de gris.

Cada una de las imágenes obtenidas las guarda en un array. Finalmente se tienen dos arrays, uno con las imágenes de entrenamiento y otro con las de test.

3. Cálculo de PCA

El objetivo de PCA es reducir el espacio de dimensiones de una imagen de d = n x m a un espacio d', analizando los componentes principales. Esta proyección no es discriminativa ya que no se tienen en cuenta las etiquetas de clase, sin embargo, haciendo esto, se conservan los ejes de mayor varianza y se consigue eliminar mucho ruido de ejes que no aportan información relevante. Esta proyección de dimensiones es una proyección lineal.

El objetivo entonces es calcular una matriz de proyección de una dimensionalidad d a una d' menor. Para ello, se ha definido un método (calculate_PCA) en el que, dadas las imágenes previamente extraídas, calcula los valores d (dimensiones), n (datos), A (matriz de datos), C (matriz de covarianzas), Δ (eigenvalues), B(eigenvectors).

El proceso de diagonalizar la matriz es muy costoso, sin embargo, si n es mucho menor que d, como es el caso (d=10304 y n=200), se puede aplicar una estrategia que reduce enormemente el tiempo de calculo de esa diagonalización. Esta técnica consiste en calcular C de la manera C' = (1/d) * A^t A en lugar de C = (1/n)AA^t .

A continuación, se diagonaliza C' haciendo uso del método linalg de la librería numpy para extraer los eigenvectores y los eigenvalues. Una vez extraídos, se ordenan para guardar aquellos cuya varianza sea mayor y finalmente se calcula la matriz B y se devuelve normalizada. Es necesaria la normalización ya que, si no, los eigenvetores serian ortogonales, pero no ortonormales, al normalizar se consigue que sean también ortonormales.

Una vez se tiene la matriz de proyección, se ha creado un método (transform) en el que, a partir del array de imágenes, la matriz de proyección y el numero de dimensiones a reducir, se transforma la imagen a las dimensiones deseadas. Para ello, extrae los eigenvectors de la matriz de proyección, tantos como dimensiones se quieran. Después, calcula la media de las imágenes y finalmente, las imágenes en la nueva dimensión de representación se calculan restando la media de las imágenes a las imágenes y multiplicando por los eigenvectors extraídos de la matriz de proyección.

Un ejemplo de una eigenface es el siguiente:

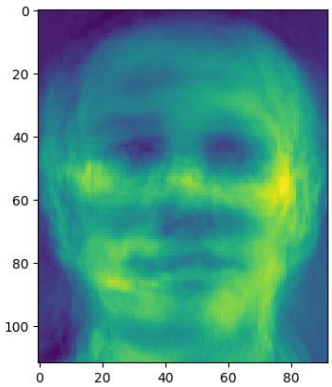
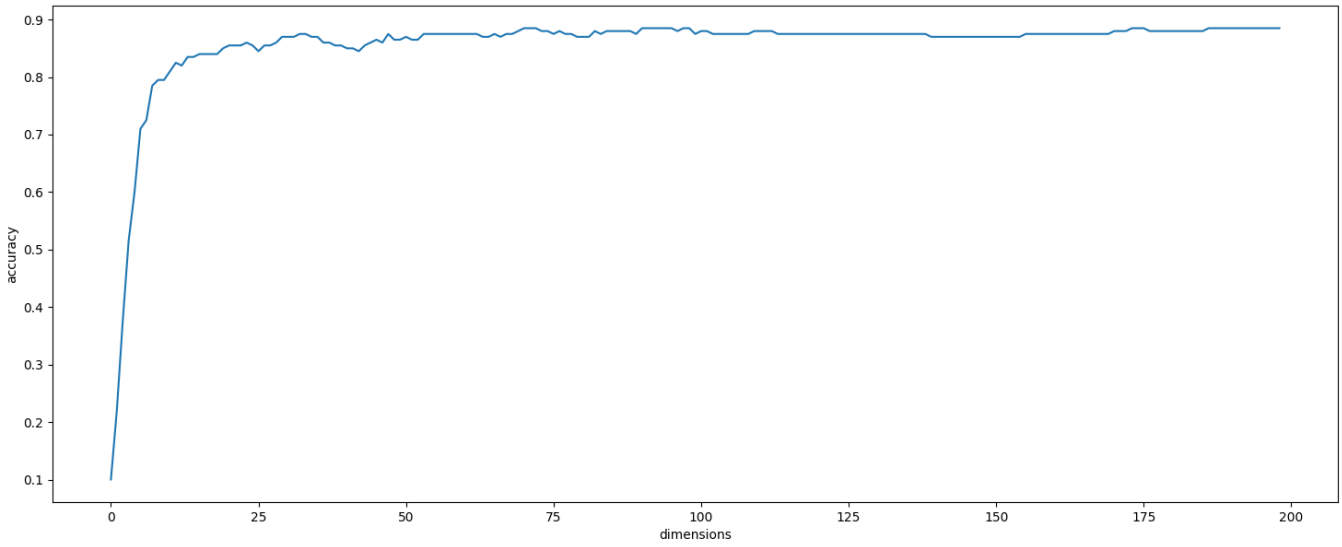


Ilustración 1. Eigenface

Este proceso de transformación se ha aplicado tanto a las imágenes de entrenamiento como a las de test. Para la clasificación, se ha usado el método del vecino más cercano. Ejecutando este script, se ha obtenido una precisión del 87% reduciendo a 30 dimensiones.

4. Variación de la dimensionalidad

En este apartado se ha estudiado como varía la precisión con diferentes tamaños de dimensión. Se ha elegido el rango de 1 a 200 para la dimensionalidad, obteniendo la siguiente gráfica:



Como puede apreciarse en la gráfica, llega un punto en el que subir la dimensionalidad no aporta nada a la precisión. Puede verse que, en torno a 70 dimensiones, subir la dimensionalidad no tiene influencia en la precisión, por lo que perfectamente se podría prescindir de ellas.

El mejor resultado en este caso ha sido obtenido con 70 componentes, obteniendo una precisión del 88.5%.

5. Conclusiones

Se ha implementado un script que calcula las eigenfaces. Se ha visto que las imágenes pueden contener dimensiones que no aportan varianza y, por tanto, no tienen influencia en la precisión del clasificador, por eso, se pueden eliminar haciendo PCA.

Además, se ha visto la problemática de la diagonalización de matrices y como ser capaces de reducir el tiempo de cómputo, siempre que se cumpla que los datos sean mucho menores que las dimensiones.

6. Bibliografía

[1] Intellipaat. How to read pgm p2 image in python, 2019