



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA



Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital
Universitat Politècnica de València

Implementación de EigenFaces

Biometría

Autor: Juan Antonio López Ramírez

Curso 2019-2020

Introducción

El objetivo de esta práctica es implementar EigenFaces partiendo de la base de datos ORL, que consiste en:

- Un conjunto de training de 200 muestras, 40 personas con 5 imágenes de cada una.
- Un conjunto de test de 200 muestras, las mismas 40 personas del conjunto de training pero con otras 5 imágenes de cada una.

Para ello se ha realizado un script en el que se realiza una reducción PCA a las muestras y, posteriormente, se entrena un clasificador del vecino más cercano.

Finalmente, se muestran los resultados del clasificador en una gráfica para comprobar como varía la precisión cuando se modifica la reducción aplicada.

Implementación y resultados

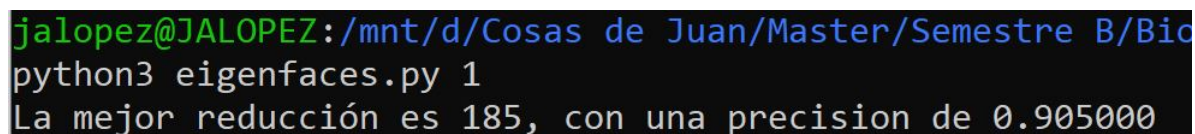
Ejecución del script

Para poder hacer uso del programa que hemos implementado, hay que ejecutar en la terminal la siguiente instrucción:

```
1 python3 eigenfaces.py v
```

Donde v es el parámetro k del algoritmo de k -vecinos más cercanos. Si le pasamos un 1, se entrenará el clasificador con el vecino más cercano, si es un 2, con los dos vecinos más cercanos, etc.

Es necesario que la carpeta ORLProcessed se encuentre en la misma ruta que el script.



```
jalopez@JALOPEZ:/mnt/d/Cosas de Juan/Master/Semestre B/Bio
python3 eigenfaces.py 1
La mejor reducción es 185, con una precision de 0.905000
```

Figura 1: Resultado de ejecutar el script realizado para 1-NN.

A continuación explicaremos como hemos obtenidos esos resultados.

Librerías utilizadas

El script se ha hecho en Python utilizando las siguientes librerías:

- **Sys**, para acceder a variables y argumentos utilizados por el intérprete que el usuario le pasa como parámetros.
- El módulo **Os**, que nos permite acceder a funcionalidades dependientes del Sistema Operativo. En nuestro caso, lo usamos para obtener información o manipular la estructura de directorios.
- El módulo **Pyplot** de la librería **Matplotlib**, para dibujar las gráficas que representan las curvas ROC.

- **Numpy**, que nos asiste a la hora de trabajar con funciones matemáticas de alto nivel y poder así operar con vectores o matrices, en caso de ser necesario. De aquí importamos también las funciones que nos aporta el módulo **linalg**, especialmente para calcular los eigenvectores y eigenvalues.
- **Scikit-learn**, para aprendizaje automático y diseñada para interoperar con numpy. De aquí obtenemos el clasificador de los k-vecinos más cercanos para no tener que implementarlo manualmente.

Procesado de datos

Primero se ha definido un método para cargar los datos.

Haciendo uso de un método obtenido de StackOverflow¹, se procesan las imágenes almacenándolas en un vector de numpy.

Hacemos esto para cada imagen y almacenamos todos los vectores en una matriz de numpy donde cada vector columna es una muestra.

De esta manera, tenemos una matriz $x_{training}$ y otra x_{test} de 10304x200 cada una (las imágenes son de 112x92=10304).

De forma simultánea, almacenamos las etiquetas de clase de training en un vector numpy ($y_{training}$) y las de test en otro (y_{test}).

Reducción de dimensionalidad con PCA

Definimos un método en python para PCA donde le pasamos los datos (entrenamiento y test) y la reducción que le queremos aplicar (d'), devolviéndonos los datos con la dimensionalidad reducida.

Primero, obtenemos las d características de las n imágenes de entrenamiento.

Luego, calculamos el vector promedio (cara promedio), que se almacenará en una matriz de numpy de tamaño $d \times 1$.

Posteriormente, calculamos la matriz A , donde almacenamos la diferencia de cada vector con el promedio. El tamaño de A será, pues, de $d \times n$.

A la hora de calcular los eigenvectores y eigenvalores, se han tenido en cuenta las consideraciones prácticas vistas en clase de teoría, en las que, en lugar de diagonalizar una matriz de covarianzas C de tamaño $d \times d$, diagonalizamos una C' de tamaño $n \times n$, obteniéndola de la siguiente forma:

$$C' = \frac{1}{d} A^t A \quad (1)$$

Cuando obtenemos sus eigenvectores (B') y eigenvalores (D'), calculamos los verdaderos de nuestro problema como:

$$B = AB' \quad (2)$$

$$D = \frac{d}{n} D' \quad (3)$$

¹<https://stackoverflow.com/questions/46944048/how-to-read-pgm-p2-image-in-python>

Como esos eigenvectores B no son ortonormales, los normalizamos dividiendo cada uno por su módulo.

Finalmente, ordenamos los eigenvectores y eigenvalores y le aplicamos a las muestras la reducción con los d' primeros eigenvectores.

Clasificador k-NN

En esta parte, definimos un método para el k-vecino más cercano donde le pasamos el parámetro k que indica el número de vecinos más cercanos y que habíamos especificado en la ejecución del script; junto con los datos y etiquetas de entrenamiento y test.

El método es un algoritmo de entrenamiento/clasificación que devuelve la precisión obtenida con los datos de entrada.

Para ello, utilizamos el método *KNeighborsClassifier* que nos proporciona scikit-learn, especificando una métrica euclídea.

Luego, entrenamos nuestro modelo con el método *fit*, al que le pasamos las muestras de entrenamiento y sus clases; y clasificamos con el método *predict*, pasándole los datos de test.

Así, obtenemos una lista de n elementos, donde cada uno de ellos indica donde se ha clasificado cada muestra. Para calcular la precisión, simplemente recorreremos esta lista y comprobamos si efectivamente se corresponden con las etiquetas de clase, obteniendo un porcentaje en tanto por uno.

Resultados

Las reducciones que hemos ido aplicando han sido desde reducir a dimensionalidad 5 hasta 200, con incrementos de 5.

Además, hemos probado desde el vecino más cercano hasta 3-NN.

De esta forma, los mejores resultados obtenidos han sido:

k-NN	Reducción (d')	Precisión (%)
1	185	90.5
2	90	85.5
3	120	86.5

En las figuras 2, 3 y 4 se puede observar como varía la precisión al modificar d' .

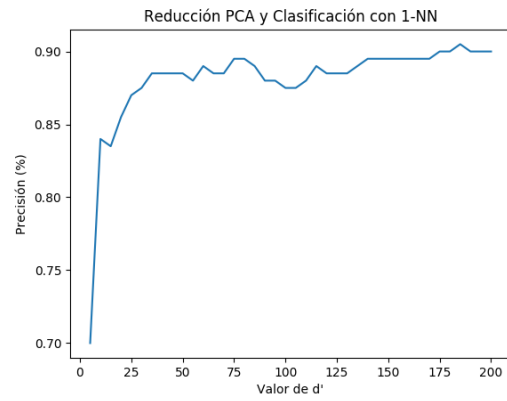


Figura 2: Resultados para el vecino más cercano.

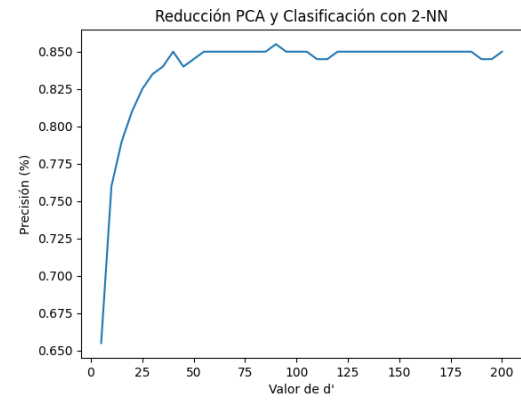


Figura 3: Resultados para 2-NN.

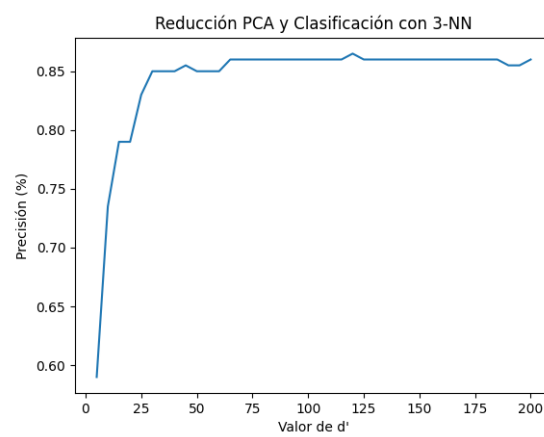


Figura 4: Resultados para 3-NN.