UNIVERSITAT POLITÈCNICA DE VALÈNCIA

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

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

Implementación de FisherFaces

Biometría

Autor: Juan Antonio López Ramírez

Curso 2019-2020

Introducción

El objetivo de esta práctica es implementar FisherFaces 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+LDA 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 LDA, con una determinada reducción PCA.

Implementación y resultados

Ejecución del script

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

```
python3 fisherfaces.py v d_pca
```

Donde *v* es el parámetro *k* del algoritmo de k-vecinos más cercanos y *d_pca* es la dimensionalidad a la que se va a reducir con PCA las muestras antes de aplicar LDA.

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

Para evitar el tener que repetir la implementación de PCA y el clasificador k-NN, hemos importado el fichero *eigenfaces.py* que habíamos desarrollado para la práctica anterior y del cuál hemos aprovechado los métodos implementados en él.

Por eso, el fichero *eigenfaces.py* se ha de encontrar en la misma ruta que nuestro script.

```
jalopez@JALOPEZ:/mnt/d/Cosas de Juan/Master/Semestre B/Biometría/Practicas/FisherFaces$
python3 fisherfaces.py 1 185
Con una reducción previa de PCA de 185, la mejor reducción con LDA es 185, obteniendo un
a precision de 0.905000
jalopez@JALOPEZ:/mnt/d/Cosas de Juan/Master/Semestre B/Biometría/Practicas/FisherFaces$
python3 fisherfaces.py 2 90
Con una reducción previa de PCA de 90, la mejor reducción con LDA es 30, obteniendo una
precision de 0.860000
jalopez@JALOPEZ:/mnt/d/Cosas de Juan/Master/Semestre B/Biometría/Practicas/FisherFaces$
python3 fisherfaces.py 3 120
Con una reducción previa de PCA de 120, la mejor reducción con LDA es 30, obteniendo una
 precision de 0.875000
```

Figura 1: Resultado de ejecutar el script realizado para 1-NN, 2-NN y 3-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.

Aunque no es una librería como tal, hemos aprovechado el script de *eigenfaces.py* e importamos los siguientes métodos:

- **read_pgm**, para almacenar una imagen en un vector de numpy.
- **procesarDatos**, en el que obtenemos dos matrices con los vectores columna que representan las muestras de training y test, respectivamente, así como las etiquetas de clase. Además, aunque no se aprovechaba en EigenFaces, este método también almacena en una variable (lista en python) el número de muestras que hay de cada clase (en nuestro caso, una lista de tamaño 40 donde todos los elementos son igual a 5, ya que tenemos 5 imágenes de cada persona).
- **PCA**. Al que se le pasa como parámetro las muestras de training y test y una dimensión d' a la que se reducen dichas muestras, y que el método retorna como salida.
- **kvecino**. Al que se le pasa como parámetro las muestras y etiquetas de clase, tanto de training como de test, junto con el parámetro v que indica si se va a realizar 1-NN, 2-NN, etc. El método retorna como salida la precisión del clasificador después de entrenar con esas muestras de training y clasificar con las de test.

Reducción de dimensionalidad con LDA

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

Primero, calculamos el vector promedio (cara promedio), que se almacenará en una matriz de numpy de tamaño dx1.

Luego, calculamos la matriz de medias por clase. Para ello, primero calculamos todos los vectores de media por clase (μ_c).

Hay un total de 40 vectores de media por clase debido a que hay 40 personas. Después, formamos la matriz de medias por clase de manera que cada vector columna está repetido en la matriz tantas veces como muestras de esa clase haya, y de forma ordenada,

por lo que la matriz será de tamaño d x n. Esto se ha realizado así para facilitarnos futuros cálculos.

Posteriormente, calculamos la matriz entre-clases Sb de la siguiente forma:

$$S_b = \sum_{c=1}^{C} n_c(\mu_c - \mu)(\mu_c - \mu)^t \tag{1}$$

Donde n_c es la media de muestras por clase. Sb es , pues, de tamaño d x d.

Luego, obtenemos la matriz intra-clases Sw de la forma que sigue:

$$S_w = \sum_{c=1}^{C} \sum_{x \in c} (x - \mu_c)(x - \mu_c)^t \tag{2}$$

Debido a que a cada muestra se le resta su correspondiente vector de clase, hemos hecho la matriz de medias por clase de tamaño d x n, donde cada vector promedio de clase se repite tantas veces como muestras de esa clase haya. El tamaño de Sw es, pues, de d x d.

Posteriormente, calculamos la matriz de covarianzas, que en este caso es el resultado de multiplicar la inversa de Sw por Sb.

Una vez hecho esto, calculamos sus eigenvalores y eigenvectores asociados, y los ordenamos.

Finalmente, aplicamos la reducción con los d' primeros eigenvectores, y retornamos como salida las muestras de train y test reducidas.

Resultados

Una vez implementado LDA y, haciendo uso de los métodos importados desde *eigenfaces.py*, procedemos a realizar la reducción PCA+LDA, entrenar el clasificador con las muestras de train reducidas y clasificar con las de test reducidas.

Para ello, dada una reducción PCA que le pasamos al script como entrada, las reducciones LDA que hemos ido aplicando han sido desde reducir a dimensionalidad 5 hasta 200, con incrementos de 5.

Además, hemos probado 1-NN, 2-NN y 3-NN.

Como en EigenFaces la mejor reducción PCA para 1-NN fue de 185, para 2-NN de 90 y para 3-NN de 120, se han mantenido esas reducciones para dichos clasificadores, variando en cada una las reducciones LDA.

Así, obtenemos como mejores resultados, variando simplemente LDA, los siguientes:

k-NN	Reducción PCA	Reducción LDA	Precisión (%)
1	185	185	90.5
2	90	30	86
3	120	30	87.5

Comparando estos resultados con EigenFaces, en los que simplemente aplicábamos PCA, podemos apreciar una mejora en la precisión de cada clasificador por separado, aunque esta sea mínima y, en el caso de 1-NN, se mantenga igual.

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

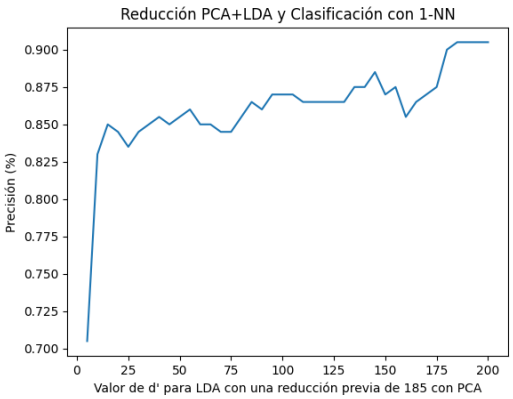

Figura 2: Resultados para el vecino más cercano.

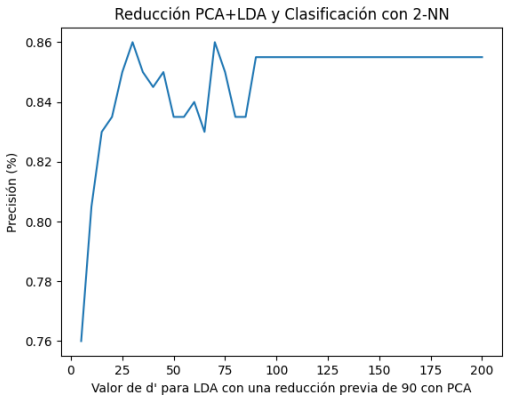

Figura 3: Resultados para 2-NN.

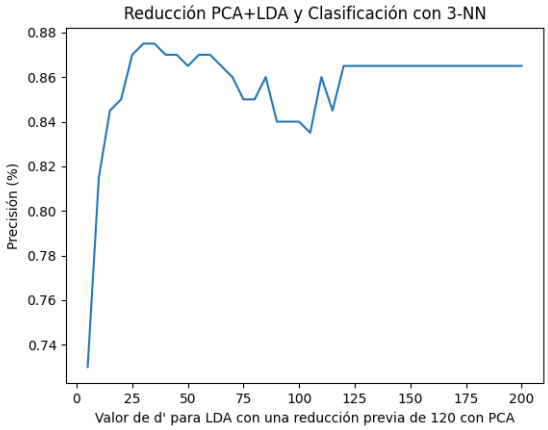

Figura 4: Resultados para 3-NN.