MEMÓRIA DE PRÁCTICAS: EIGENFACES

Biometría – Master en IARFID

Enric Bonet Cortés
Universidad Politécnica de Valencia

Implementar Eigenfaces:

- Probar con ORL:
  - Training: imágenes [1 – 5] de cada individuo
  - Test: imágenes [6 – 10] de cada individuo
- Utilizar el vecino más cercano
- Obtener curvas de error variando d'

Para este segundo ejercicio, el dataset se no proporciona a partir de un enlace de *Dropbox*. Para poder utilizar, lo primero que se ha hecho ha sido descargar este dataset del enlace de *Dropbox*, y almacenarlo en *Google Drive* comprimido en formato **ZIP**. Esto se ha hecho por motivos de comodidad, ya que en este caso el problema se ha resuelto utilizando *Google Collaboratory*, el cual nos proporciona una librería que interacciona con el servicio de *Drive* para montar la ruta de las carpetas que tengamos almacenadas en Drive dentro de cualquier Notebook de *Collab*.

Tras esto, puesto que las imágenes se nos proporcionan en formato **PGM** se ha recurrido al siguiente enlace de *StackOverflow* para poder realizar una lectura correcta de las imágenes:

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

Una vez implementada una función similar a la que contiene el enlace anterior, otra función llamada **get_images** se ocupa de realizar la búsqueda y lectura de las imágenes, separando estas en cada corpus (train y test) tal y como manda el enunciado, almacenándolas en objetos del tipo Array de la librería **Numpy** de *Python*. Esta conversión en Arrays, permite posteriormente pasar estas imágenes a objetos de tipo Matrix de la misma librería, los cuales no permiten operar a alto nivel con matrices de datos y recrear de esta forma las fórmulas descritas en las diapositivas de la asignatura, lo mas cómodamente posible.

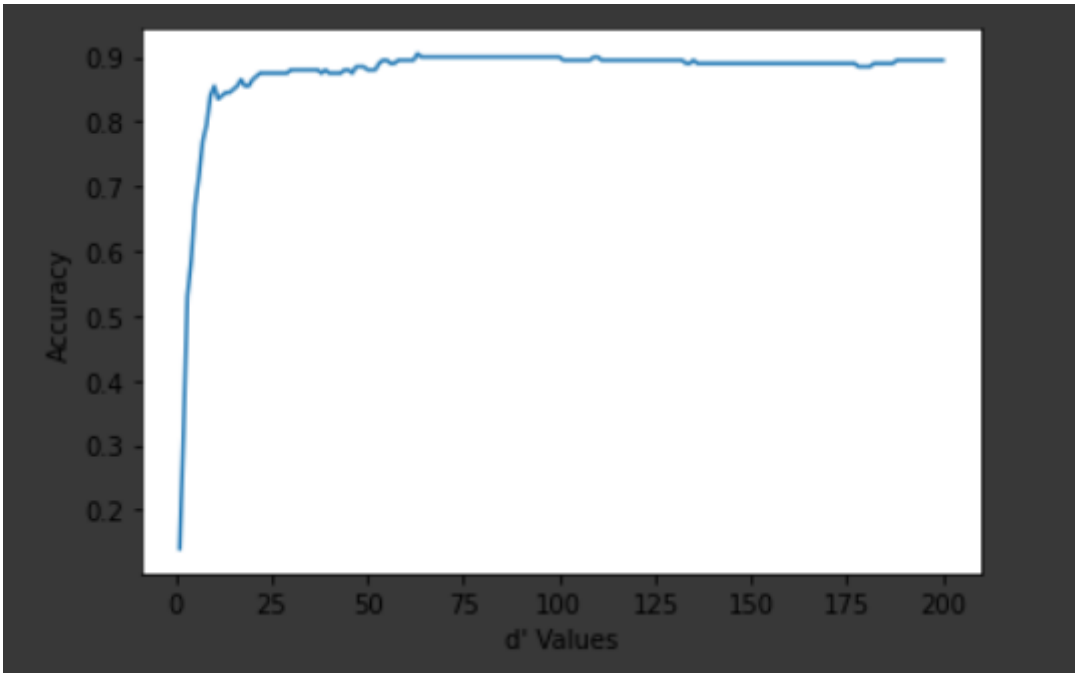
Se ha construido también, después de esta conversión de datos, una función llamada **PCA**, que se encarga de devolver los **EigenVectores** de la matriz de datos que se le proporcione como parámetros. Esta función, utiliza diversas utilidades de la estructura de datos Matrix para calcular de forma rápida y sencilla propiedades de las matrices como la transpuesta, la media de los datos, o la inversa de la matriz (**T, mean(x=1)** y **I** respectivamente), y así también calcular la matriz de covarianzas que describen las diapositivas (**C**) en el apartado de ***Consideraciones Prácticas***.

Una vez obtenida esta matriz, se calcula sus EigenVectores y EigenValues con la función **eigh** de **Numpy.lianlg**. Estas dos listas que nos devuelve el método anterior, corresponden con las variables **B'** y **Delta'** descritos en Consideraciones Prácticas, así que una vez son devueltos por la función eigh, se calculan los verdaderos EigneVectores y EigenValues del conjunto de datos para, posteriormente, ordenar estos EigenVectores por medio del método **sorted** de Python, según la lista de EigneValues. Para ello junto al método **sorted,** se utiliza la función **enumerate** sobre los EigenValues para ordenar una lista de tuplas de "índice-EigenValue", que será ordenada por su EigenValue. De esta forma, luego se utiliza esta lista para saber con que

índices se deben de ordenar los EigenVectores, para finalmente ser devueltos por la función PCA**.**

La última función implementada es **reduce**, la cual se encarga de trasformar el corpus original en un corpus reducido aplicando los EigenVectores devueltos por la anterior función y un número de componentes que se le pase también como parámetro, que corresponde con la variable **d'** de las diapositivas.

Finalmente, se ha generado un bucle que permite probar todos los valores de d' posibles (200 para este problema) sobre la reducción PCA implementada y que imprime los valores de precisión obtenidos para cada d' posible sobre el clasificador "Vecino Mas Cercano" (1-NN) de la librería **Sklearn** de Python. A su vez, este bucle almacena estos valores en dos listas que, más tarde, dibujaran una gráfica a través de la librería **Matplotlib.pyplot** que muestra la curva de precisión obtenida por este problema sobre EigenFaces.



La mejor precisión obtenida por esta implementación se encuentra con un valor de d' = 63 y corresponde a un 90,5% de acierto. En la gráfica anterior se puede verificar observando un pequeño pico en la curva entorno a este valor de d'.