Victoria Beltrán Domínguez

Memoria prácticas biometría

En esta memoria se pretende describir brevemente los ejercicios realizados para las prácticas de la asignatura.

1. **Curva ROC (2.5 puntos)**

   Implementar un programa que dado dos ficheros con los scores clientes y scores impostores obtenga:
   - Curva ROC
   - FP (FN = X) y umbral
   - FN (FP = X) y umbral
   - FP = FN y umbral
   - Área bajo la curva ROC
   - D-prime

Para este primer ejercicio se ha entregado el archivo *roc_curve.py*
Con el fin de facilitar la ejecución se han definido diferentes argumentos:

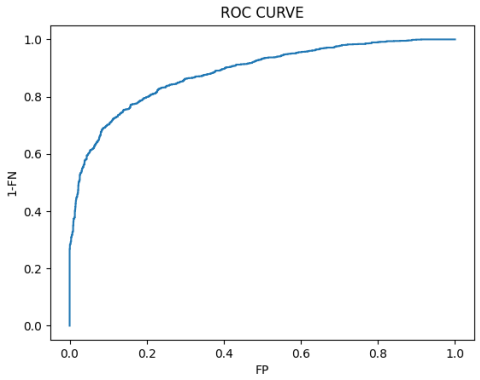
- -c (obligatorio): especificación del archivo con los scores de clientes.
- -i (obligatorio): especificación del archivo con los scores de impostores.
- -fn (opcional): valor específico de falsos negativos para los que se quiere encontrar el correspondiente valor de falsos positivos. Este argumento sólo será necesario cuando queramos calcular FP (FN = X) y su umbral.
- -fp (opcional): valor específico de falsos positivos para los que se quiere encontrar el correspondiente valor de falsos negativos. Este argumento sólo será necesario cuando queramos calcular FN (FP = X) y su umbral.

Básicamente, para dibujar la curva ROC, lo que se ha hecho es navegar por los diferentes thresholds dados, e ir de manera incremental guardando los diferentes puntos de la curva.
De esta manera, primero se han especificado los valores cuando el threshold es 0, y a partir de ahí se ha ido cogiendo el menor threshold de las listas de clientes e impostores, sumando 1 a FN si el threshold mínimo corresponde con un cliente, restando 1 a FP si el threshold mínimo corresponde a un impostor y haciendo ambas si ambos thresholds tienen el mismo valor. También debemos actualizar los índices cada vez e ir guardando los puntos obtenidos (normalizando sus valores para poder obtener directamente la curva ROC).

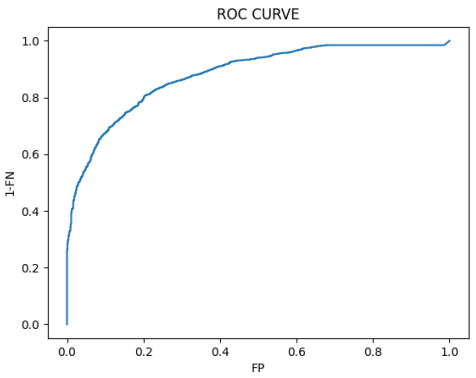
Durante este proceso en el que vamos comparando thresholds e incrementando/decrementando las variables, también vamos a calcular cuándo FP y FN son iguales (o muy cercanos), y en caso de que nos lo pidan, también se calcularán los FP (FN = X) y FN (FP = X) con sus respectivos umbrales.

Finalmente, con el fin de calcular el área bajo la curva ROC se ha utilizado el método de Mann-Whitney U test, pues se había probado otro con numpy pero era más costoso. Finalmente, para calcular el valor D-prime simplemente se ha implementado la fórmula especificada en las diapositivas.

Especificando como entrada al sistema los ficheros de scoresA y scoresB respectivamente, obtenemos las siguientes curva ROC:



Curva ROC para scoresA



Curva ROC para scoresB

2. Implementar eigenfaces (4 puntos)

Probar con ORL:

Training: imágenes [1 − 5] de cada individuo.

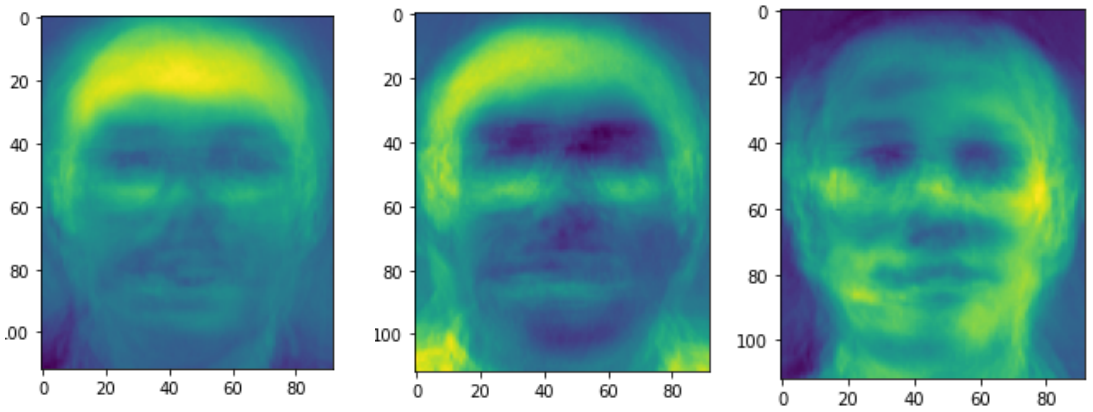
Test: imágenes [6 − 10] de cada individuo.

Utilizar el vecino más cercano.

Obtener curvas de error variando d.

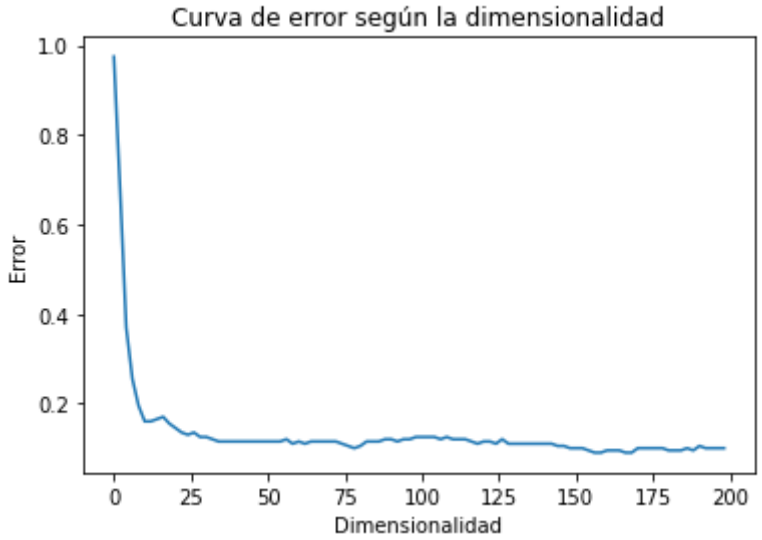
Se valorará haber tenido en cuenta las consideraciones prácticas.

Para este ejercicio se ha entregado el archivo *PCA_LDA.ipynb*. En este archivo solucionaremos tanto este ejercicio como el siguiente. Lo primero que hacemos es cargar los datos y guardar cada imagen como una columna en la matriz (X si es de entrenamiento y Y si es de test). A continuación calculamos la matriz A, restando a cada muestra de X su media, y a partir de A podemos calcular C. Una vez calculado, se deberá resolver la ecuación $CB = B\Delta$, diagonalizando C. Como normalmente la dimensión de C es realmente grande, para facilitar el problema solucionamos directamente C' (nxn en vez de dxd) y obtenemos B y Δ realizando las operaciones matemáticas indicadas en las transparencias de la asignatura (dividiendo los eigenvectores por sus módulos para que sean ortonormales). Para asegurar que estas operaciones son correctas, se han sacado diferentes EigenFaces, tal y como podemos ver en las siguientes imágenes:



Una vez tenemos los eigenvectores simplemente proyectamos X a distintas dimensiones, y para el test proyectamos y miramos que muestra de X se queda más cerca para clasificarla en esa persona.

Probando diferentes dimensiones, obtenemos que el mejor resultado para PCA tiene una precisión de 0.905 proyectando a 150 dimensiones. También se ha sacado la curva del error variando la dimensionalidad y la podemos encontrar en la siguiente figura:



3. Implementar fisherfaces (5 puntos)

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.

Se valorará haber tenido en cuenta la estabilidad numérica.

Para este tercer ejercicio se ha entregado el archivo *PCA_LDA.ipynb*. La manera de cargar los datos ha sido igual que en el anterior.
En este caso, sin embargo, lo primero que hacemos es sacar las matrices de dispersión intra e inter clase (Sb y Sw). Vemos que este cómputo empieza a hacerse pesado, así que decidimos aplicar PCA antes de computar LDA. Así, los datos de partida serán los datos originales proyectados con PCA (a una dimensión fija *d* que va variando).
Una vez lo tenemos, debemos resolver Sb B = Sw B Δ. Utilizamos la función eig de scipy.linalg, que se usa para problemas de eigenvectores generalizados.
Una vez obtenemos los eigenvectores hacemos lo mismo que hicimos para el anterior ejercicio con el fin de obtener accuracy para test.

Probando diferentes dimensiones, obtenemos que el mejor resultado tiene una precisión de 0.92 proyectando inicialmente a PCA 200 y posteriormente proyectando esos datos con LDA 37.

También se ha sacado la curva del error variando la dimensionalidad tanto de PCA como de LDA: