Curso 2021-2022

Iñaki Diez Lambies y Manuel Diaz

Percepción

Proyecto de prácticas

Reconocimiento de dígitos manuscritos: MNIST

Contenido

[Ejercicio 2.1 - Principal Component Analysis 2](#_Toc101787255)

[Ejercicio 2.2 – Comprobación PCA 2](#_Toc101787256)

[Ejercicio obligatorio – KNN + PCA 2](#_Toc101787257)

[pca+knn-exp.py 2](#_Toc101787258)

[pca+knn-eva.py 3](#_Toc101787259)

[Ejercicio opcional 3](#_Toc101787260)

[Algoritmo de Wilson 3](#_Toc101787261)

Entrega 1

# Ejercicio 2.1 - Principal Component Analysis

Para este ejercicio se nos ha pedido realizar una implementación del algoritmo de Principal Component Analysis (PCA). En nuestro caso, la función que designaremos tiene como parámetro de entrada los datos de entrenamiento dispuestos por filas y, como resultado, no devolverá dos elementos: el vector media de los valores (media entre todas sus dimensiones) y la matriz de proyección *W*. Esta matriz se compone por los vectores propios de la matriz de entrada dispuestos en filas y ordenados de mayor a menor valor propio asociado.

## Ejercicio 2.2 – Comprobación PCA

Se nos pide comprobar el correcto funcionamiento del algoritmo anterior. Para esto hemos desarrollado un pequeño script que nos permite realizar el visionado de los vectores propios de los datos dados.

Para conseguir esto primero aplicamos PCA a los datos y seguidamente realizamos la representación vector propio a vector propio siguiendo lo realizado en la práctica 0. El script está diseñado para mostrarlos todos, de forma que para no tener que observar todo el rango de la matriz se debe parar la ejecución del programa a través de línea de comandos.

# Ejercicio obligatorio – KNN + PCA

Este ejercicio consiste en estudiar el comportamiento del clasificador por *k* vecinos más cercanos (KNN) respecto a la redimensionalidad aplicada a la matriz de muestras de entrenamiento. Esta última se consigue a partir de aplicar las *k* columnas de la matriz de proyección *W* a las muestras de entrenamiento (para *k* dimensiones, *k* columnas).

Vamos a completar dos programas para experimentar con esto.

## pca+knn-exp.py

Este programa nos permite designar, a partir de un conjunto de muestras de entrenamiento, una parte dedicada al entrenamiento y otra a realizar los test. Además, podemos designar un conjunto de dimensionalidades a aplicar y que, gracias a la función *knn*, podemos obtener el error de clasificación por vecinos más cercanos para unas muestras de entrenamiento y test dadas.

Así pues, nuestro programa separa de un conjunto de muestras en entrenamiento y test (líneas 23-27) después de ser aleatorizadas las muestras (líneas 19-21). Seguidamente realizamos el cálculo de la matriz de proyección para, a continuación, calcular la tasa de error para cada posible proyección en *k* dimensiones.

Siguiendo las directrices del boletín, y con el objetivo de replicar la gráfica dada, realizamos un experimento con el 90% de las muestras para entrenamiento y el 10% para test, así como pruebas con redimensionalidad en 1, 2, 5, 10, 20, 50, 100, 200 y 500 dimensiones.

Estos resultados son anotados siguiendo el formato indicado para realizar después la representación con GNUPlot. El resultado de esto se puede visualizar en el archivo *pca+knn-exp.eps*.en forma de dibujado vectorial. También se pueden consultar los resultados en el fichero *pca+knn-exp.out*.

## pca+knn-eva.py

Ahora calculamos gracias a este script la tasa de error para todas las muestras de los datos MNIST. En concreto utilizaremos la dimensionalidad que, según nuestros resultados, aporta una tasa de error incluso que la original. Esta es 50 dimensiones.

Así pues hemos realizado un experimento muy similar solo que para una dimensión y aplicando esta vez todo el conjunto de muestras de entrenamiento y las muestras de test de MNIST. Como resultado esto nos ha dado un error de .

# Ejercicio opcional

## Algoritmo de Wilson

Hemos escogido implementar este ejercicio siguiendo las recomendaciones dadas en el anexo. Así pues, en el archivo *wilson.py* hemos realizado la implementación de los tres métodos descritos en el boletín.

En primer lugar, realizamos *mnn*. Esta función tiene como objetivo que, dada una matriz con los datos de entrenamiento por filas *X*, un vector columna *xl* con las etiquetas de estos y el número *m* de vecinos más cercanos que queremos almacenas; calcular la matriz *V* que almacena por columnas los índices de los *m* vecinos más cercanos de cada prototipo (fila) de *X*.

El funcionamiento de esta es el siguiente:

1. Inicializamos la matriz *V* a una matriz de ceros con la dimensionalidad deseada (*m*, *nº prototipos*).
2. Para cada muestra realizamos el calculo de la distancia L2 adaptando una de las implementaciones dadas en el documento *L2dist.py* a nuestro calculo iterativo.
   1. Después de este cálculo utilizamos la función *argsort* para conseguir los índices de las distancias más pequeñas a la muestra en cuestión.
   2. Guardamos los *m* resultados más cercanos en la matriz *V*. Esto lo hacemos obviando el elemento más cercano (distancia 0, una muestra consigo misma).
3. Después de realizar el cálculo devolvemos *V* como resultado.

[Justificación de porqué hemos realizado esto muestra a muestra y no todo de una con L2dist original]

Seguidamente realizamos la implementación del método *knnV*. Este nos permite devolver la clase a la cual pertenece el prototipo *i*. Esto lo hace a partir de la columna con los índices de los prototipos más cercnoas *Vi*, el conjunto de índices aún disponibles de Wilson *ind*, las clases de las muestras *xl* y el número de vecinos más cercanos a tener en cuenta *k*.

El funcionamiento de este es el siguiente:

1. Filtramos en *idx* los índices de *Vi* que aún no han sido eliminados
2. Escogemos los *k* primeros a considerar
3. Realizamos la clasificación y la guardamos en *c* (basándonos en el código de *knn.py*)
4. Devolvemos el resultado

Por último la función *wilson* se encarga de utilizar