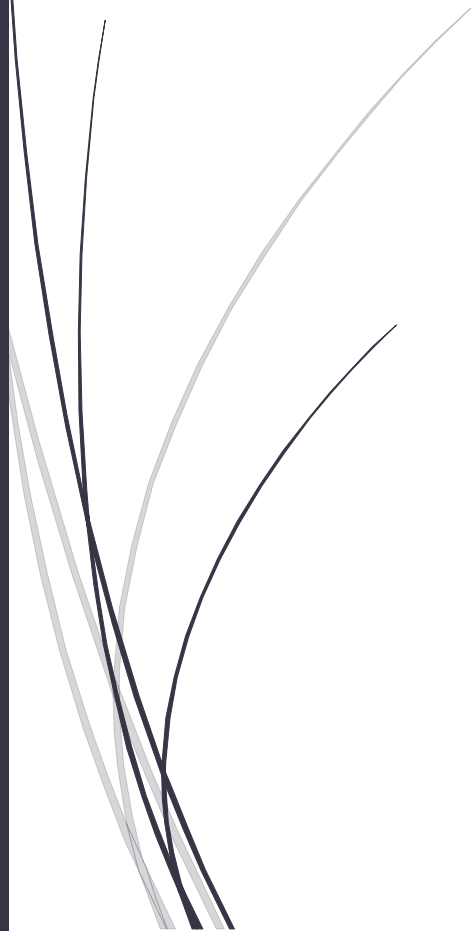
Curso 2021-2022

Proyecto de prácticas

Reconocimiento de dígitos manuscritos: MNIST

Parte obligatoria



Iñaki Diez Lambies y Manuel Diaz Pastor
PERCEPCIÓN

1 CONTENIDO

Entrega 1

2 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.

2.1 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.

3 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.

3.1 PCA+KNN-EXP.PY

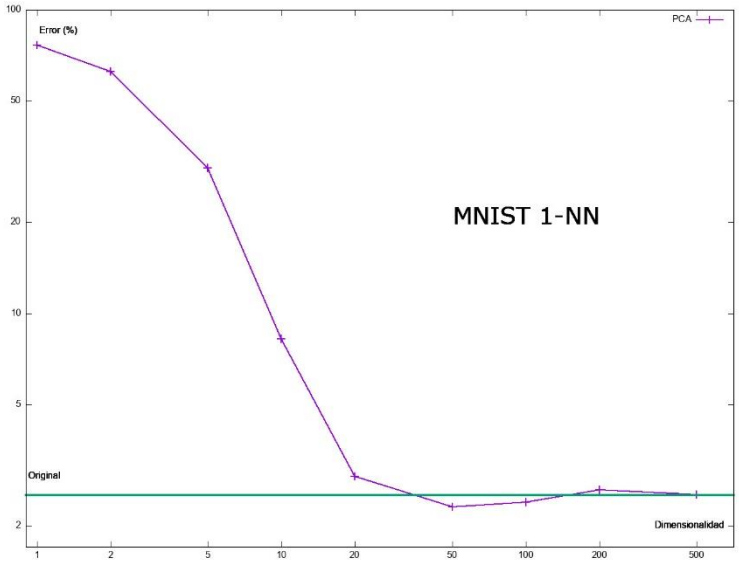
Este programa nos permite designar, a partir de un conjunto de muestras de entrenamiento, una parte dedicada al entrenamiento y otra a realizar los test. Además, podemos designar un conjunto de dimensionalidades a aplicar y que, gracias a la función *knn*, podemos obtener el error de clasificación por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

3.1.1 Resultados



Como podemos observar en la imagen, para una baja dimensionalidad el error excede en gran medida el del modelo original. Ahora bien, conforme aumentamos la dimensionalidad encontramos una mayor cercanía al punto de partida.

Incluso podemos encontrar un conjunto de dimensionalidades óptimas que la original, donde el error es inferior. En nuestro caso hemos obtenido que para 50 dimensiones el error se reduce a un 2.3%.

3.2 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 2.6597%.

3.2.1 Resultados

Si comparamos este resultado con los valores presentados en la página oficial de MNIST, podremos comprobar que la tasa de error obtenida mejora significativamente a la de clasificadores lineales. También podemos observar una leve mejoría con respecto a clasificadores de k vecinos más cercanos con distancias L2 y L3.

CLASSIFIER	PREPROCESSING	TEST ERROR RATE (%)
Linear Classifiers		
linear classifier (1-layer NN)	none	12.0
linear classifier (1-layer NN)	deskewing	8.4
pairwise linear classifier	deskewing	7.6
K-Nearest Neighbors		
K-nearest-neighbors, Euclidean (L2)	none	5.0
K-nearest-neighbors, Euclidean (L2)	none	3.09
K-nearest-neighbors, L3	none	2.83
K-nearest-neighbors, Euclidean (L2)	deskewing	2.4

A partir de varias técnicas de preprocesado que se han llevado a cabo en los ejemplos aquí expuestos, los resultados que hemos obtenidos resultan peores en comparación, quedando por debajo de la mayoría de redes neuronales y máquinas de vectores soporte (SVMs), pero ligeramente mejor que ciertos clasificadores no lineales (cuadráticos y RBF):

Non-Linear Classifiers		
40 PCA + quadratic classifier	none	3.3
1000 RBF + linear classifier	none	3.6