Doble Grado en
Ingeniería Informática y Matemáticas

APRENDIZAJE AUTOMÁTICO
(E. Computación y Sistemas Inteligentes)

TRABAJO-3: Programación
AJUSTE DE MODELOS LINEALES

UNIVERSIDAD DE GRANADA

Carlos Santiago Sánchez Muñoz

Grupo de prácticas 3 - Lunes

Email: carlossamu7@correo.ugr.es

29 de mayo de 2020

Índice

1. Clasificación

El primer problema a abordar es un problema de clasificación. La base de datos es a usar es *Optical Recognition of Handwritten Digits* que contiene imágenes de dígitos del sistema de numeración arábigo. El objetivo consiste en aprender de esta base de datos para poder clasificar otras imágenes con dígitos manuscritos.

1.1. Comprensión del problema a resolver

El conjunto de datos lo obtenemos en dos ficheros sacados de la dirección proporcionada en el guión:

- `optdigits.tra` para el conjunto de entrenamiento.
- `optdigits.tes` para para el conjunto de test.

Ambos conjuntos de datos son una batería de imágenes de dígitos manuscritos con un preprocesamiento previo. Estos ficheros tienen un formato CSV (*Comma-Separated Values*) lo cual significa que las filas están separadas por saltos de línea y las columnas por el símbolo ',' (coma).

Vamos a explicar la técnica de reducción de la dimensionalidad que se ha aplicado a cada dígito para obtener su vector de características. Los dígitos son un *bitmap* de tamaño 32×32 en donde cada píxel es blanco o negro. Se han dividido en bloques de tamaño 4×4 no superpuestos. El vector de características son 64 valores del intervalo $[0, 16]$ que indican el número de píxeles negros de cada uno de los 8×8 bloques de tamaño 4×4. La etiqueta de cada dato es el dígito al que se corresponde, un entero en $[0, 9]$.

El problema a resolver es un problema de aprendizaje supervisado. La matriz $\mathcal{X}$ es la matriz de características que en cada fila tendrá los 64 valores ya explicados y tantas filas como instancias. El vector de etiquetas $\mathcal{Y}$ está compuesto por enteros en $[0, 9]$ y su dimensión es el número de instancias. La función de objetivo es $f : \mathcal{X} \to \mathcal{Y}$ que para cada instancia $(x_n, y_n) \in \mathcal{X} \times \mathcal{Y}$ verifica $f(x_n) = y_n$.

1.2. Selección de las clase/s de funciones a usar

En la búsqueda de la función objetivo f es necesario fijar la clase de funciones o conjunto de hipótesis $\mathcal{H}$. Más adelante elegiremos $g \in \mathcal{H}$ de modo que $g \approx f$.

La clase de funciones elegida es la de combinaciones lineales de los datos. Con las dimensiones que se manejan en este problema usar una clase no lineal puede ser problemático y dar lugar a *overfitting*. Además carecería de sentido pues qué significa elevar al cuadrado el número de unos que hay en un bloque 4×4. A priori no parece que eso funcionase así que nuestra clase elegida es:

$$\mathcal{H}_{clas} = \left\{ w_0 + \sum_{i=1}^{N} w_i x_i, \quad w \in \mathbb{R}^{N+1} \right\}.$$

1.3. Fijando los conjuntos de training y test

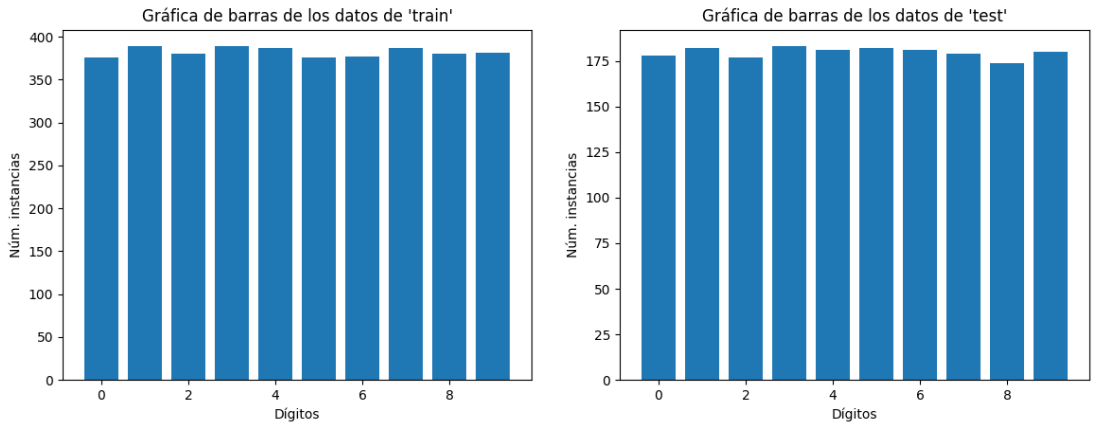
En este caso el conjunto de datos proporcionado venía separado distinguiendo el conjunto de entrenamiento del conjunto de test. Vamos a limitarnos a estudiar y averiguar algunos datos de este reparto para decir si es lo que deseamos.

El número de instancias del conjunto de entrenamiento es 3823 y el de test es 1797. Están repartidos por tanto con un porcentaje 68,025 % y 31,975 % respectivamente. Estos datos son muy razonables, falta saber si el porcentaje de aparición de cada dígito está bien repartido en cada conjunto de entrenamiento. En la Tabla 1.

Tabla 1: Reparto de los dígitos en 'train' y 'test'

Díg.	**Instancias 'train'**	**Porcentaje 'train' (%)**	**Instancias 'test'**	**Porcentaje 'test' (%)**
0	376	9.84	178	9.91
1	389	10.18	182	10.13
2	380	9.94	177	9.85
3	389	10.18	183	10.18
4	387	10.12	181	10.07
5	376	9.84	182	10.13
6	377	9.86	181	10.07
7	387	10.12	179	9.96
8	380	9.94	174	9.68
9	382	9.99	180	10.02

En unas gráficas de barras lo podemos apreciar mejor:



(a) Train (b) Test

Imagen 1: Gráficas de barras de los datos

En efecto, las apariciones está bien estratificadas y dicha uniformidad es necesaria para el buen aprendizaje del algoritmo que vamos a construir.

1.4. Preprocesado de los datos

Antes de hacer el preprocesado vamos a verificar algunas cosas que debe de cumplir la base de datos después de la comprensión y estudio realizado. En primer lugar se ha comprobado la no existencia de valores perdidos u *outliers*. En un base de datos tan conocida era obvio que esto no iba a pasar. A continuación se ha comprobado que todos los valores sean enteros. Por último, todos los valores de las características deben de estar en el rango $[0, 16]$ y los de las etiquetas en $[0, 9]$. Veamos estas comprobaciones:

```
Outliers en 'train': 0
Outliers en 'test': 0
Todos los valores son enteros en 'train': True
Todos los valores son enteros en 'test': True
Intervalo en el que están las características de 'train': [0,16]
Intervalo en el que están las etiquetas de 'train': [0,9]
Intervalo en el que están las características de 'test': [0,16]
Intervalo en el que están las etiquetas de 'test': [0,9]
```

Imagen 2: Comprobaciones sobre los datos

El preprocesamiento de los datos es muy importante para el buen ajuste del modelo. Es sin duda un paso fundamental ya que previere del sobreajuste a los datos y de la redundancia a la vez que reduce la dimensionalidad del problema haciéndolo menos complejo y más eficiente en tiempo. Para ello después de diferentes pruebas se ha procedido a tres pasos:

- Eliminar aquellas características cuyo valor sea constante. Para ello simplemente se eliminan aquellas que tienen varianza cero lo cual es equivalente. En este paso se usa `VarianceThreshold` de `sklearn.feature_selection`.
- Análisis de Componentes Principales PCA (Principal Component Analysis). Esta técnica reduce la dimensionalidad del problema sin perder mucha capacidad para explicar la varianza de los datos. El espacio de trabajo se transforma mediante esta técnica a uno más pequeño conservando un porcentaje de varianza explicada elegido, en nuestro caso un 95 %. Es probable que algunas zonas de la imagen no sean tan importantes y con esta técnica las podamos descartar. Para este paso usamos `StandardScaler` de `sklearn.preprocessing`.
- Para poder aplicar PCA los datos deben de tener una distribución gaussiana. Por consiguiente se van a normalizar para que la media sea 0 y la varianza 1. Para este paso se importa `PCA` de `sklearn.decomposition`.

Estos tres pasos los unimos con `Pipeline` disponible en `sklearn.pipeline`. Básicamente dados unos datos en cada paso se hace el `fit_transform` a ellos y se pasa al siguiente elemento del `Pipeline`.

```
Número de características de 'train' antes del preprocesado: 64
Número de características de 'train' después del preprocesado: 41
```

Imagen 3: Número de características antes y después del preprocesamiento

Tal y como vemos los resultados del preprocesamiento son un decremento considerable en el número de características. Además, voy a pintar las matrices de correlación antes y después del preprocesamiento.

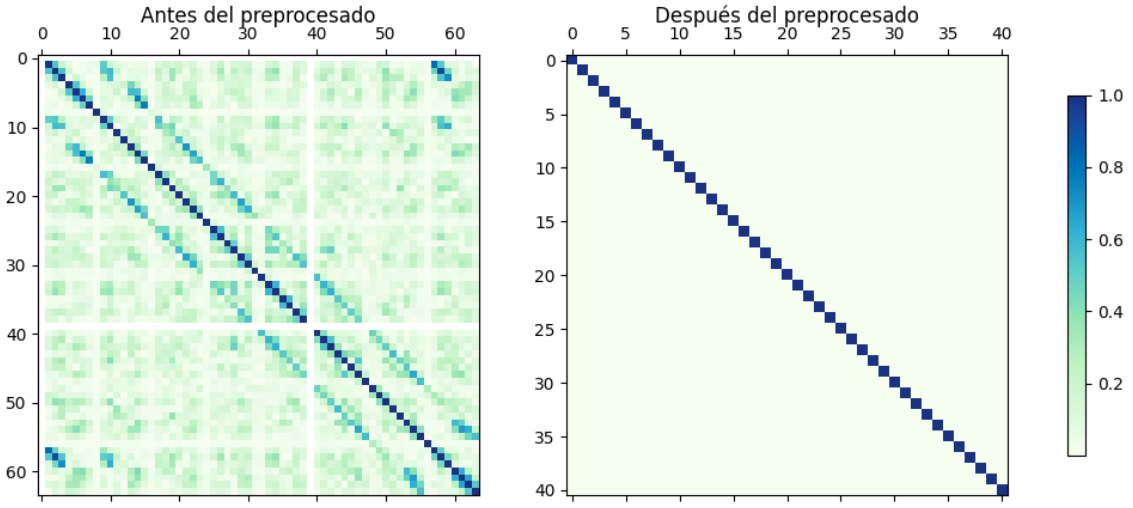


Imagen 4: Matriz de correlación antes y después del preprocesado

La columna y fila de píxeles en blanco en la matriz de correlación antes del preprocesamiento indica que esos valores son constantes (los elimina `VarianceThreshold`).

1.5. Fijando la métrica de error a usar

El error que voy a usar es el porcentaje de acierto en la clasificación de los elementos. Este error se denomina *accuracy* y así lo indicaremos a los modelos de `sklearn`.

Es necesario justificar esta elección porque no siempre sería acertada. Si mi base de datos estuviese desbalanceada y hubiese un alto porcentaje de instancias de una misma clase es probable que el error *accuracy* fuese en general bueno aunque el modelo no fuese el idóneo. Sin embargo, aquí los datos están repartidos para cada dígito de manera equitativa.

1.6. Discusión de la técnica de ajuste elegida

Texto.

1.7. Discusión de la necesidad de regularización

En su caso la justificar la función usada para ello.

1.8. Identificación de los modelos a usar

Texto.

1.9. Estimación de hiperparámetros y selección del mejor modelo

Texto.

1.10. Estimación por validación cruzada de E_{out} y comparación con E_{test}

Texto.

1.11. Modelo a proponer a la empresa

Suponga que Ud ha sido encargado de realizar este ajuste para una empresa. ¿Qué modelo les propondría y que error Eout les diría que tiene?. Justifique las decisiones.

2. Regresión

2.1. Comprensión del problema a resolver

Texto.

2.2. Selección de las clase/s de funciones a usar

Texto.

2.3. Fijando los conjuntos de training y test

Texto.

2.4. Preprocesado de los datos

Codificación, normalización, proyección, etc. Es decir, todas las manipulaciones sobre los datos iniciales hasta fijar el conjunto de vectores de características que se usarán en el entrenamiento..

2.5. Fijando la métrica de error a usar

Texto.

2.6. Discusión de la técnica de ajuste elegida

Texto.

2.7. Discusión de la necesidad de regularización

En su caso la justificar la función usada para ello.

2.8. Identificación de los modelos a usar

Texto.

2.9. Estimación de hiperparámetros y selección del mejor modelo

Texto.

2.10. Estimación por validación cruzada de E_{out} y comparación con E_{test}

Texto.

2.11. Modelo a proponer a la empresa

Suponga que Ud ha sido encargado de realizar este ajuste para una empresa. ¿Qué modelo les propondría y que error Eout les diría que tiene?. Justifique las decisiones.

```
def ejercicio3_2():
```