

PRÁCTICA 3: Ajuste de Modelos Lineales

*Aprendizaje Automático || Curso 2018-2019*

**Alumno:** José María Sánchez Guerrero

**DNI:** 76067801Q

**Correo:** jose26398@correo.ugr.es

**Grupo:** A3 – Viernes 17:30

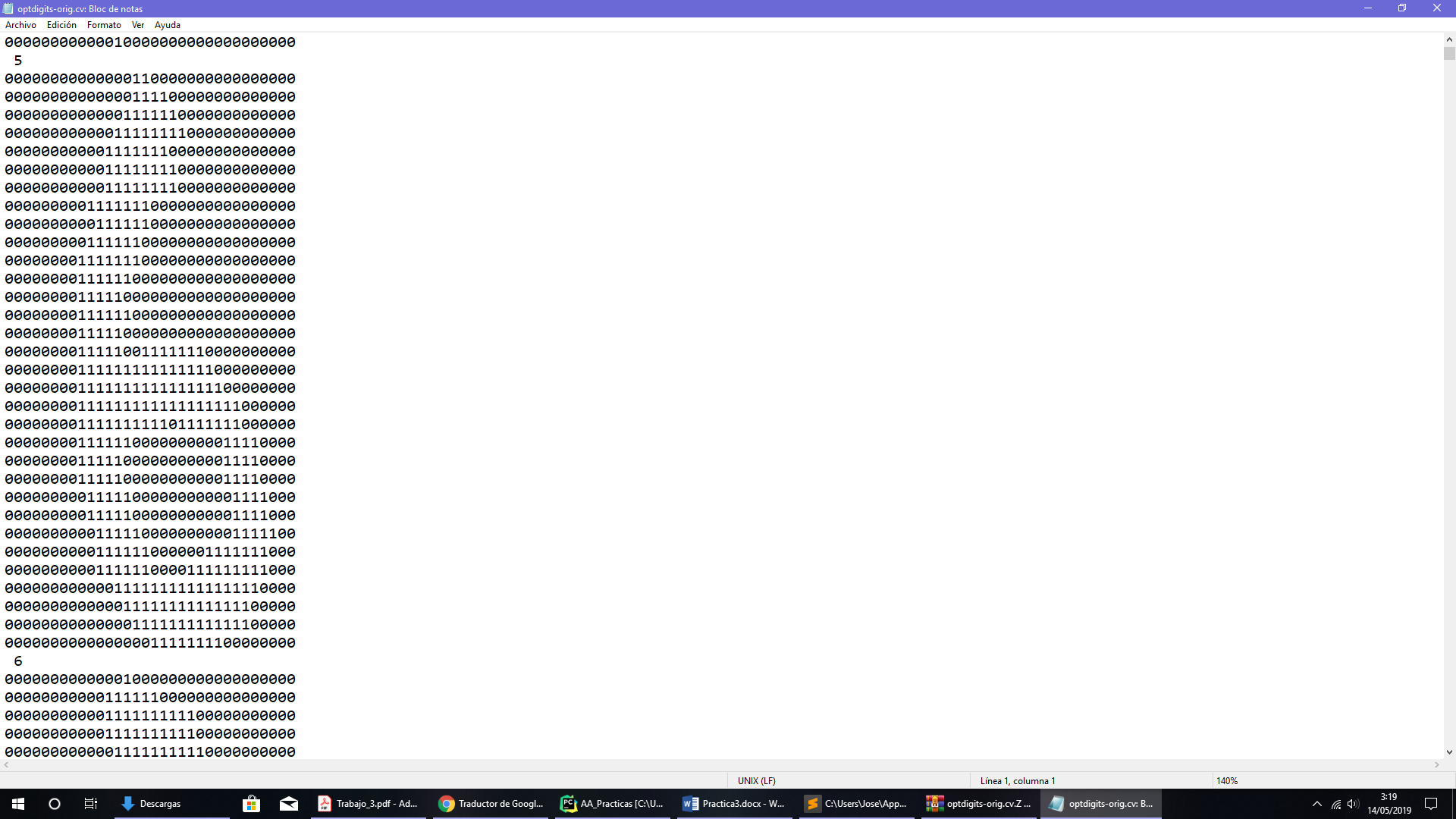
# 1. Comprender el problema a resolver

El objetivo de esta práctica será ajustar y seleccionar el mejor modelo lineal para un problema determinado, y posteriormente discutir los resultados (y el error) de éste modelo elegido. Para llevarlo a cabo, se utilizarán dos conjuntos de datos sacados del *“UCI Machine Learning Repository”*.

## *Optical Recognition of Handwritten Digits Data Set*

Este conjunto de datos está formado por una serie de filas que representan un número y 65 columnas, formadas por 64 características de un número entero (del 0 al 9) que estará en la última columna de cada fila.

Los datos originalmente son mapas de bits normalizados de dígitos manuscritos de un formulario preimpreso. Estos mapas son de tamaño 32x32 y cada uno de los bits estará activo o no, dependiendo de si en el formulario está pintado o no. Estos mapas de bits se dividirán en bloques no superpuestos de 4x4 y el número de bits activos en este bloque será el número que encontramos en nuestro dataset. Al ser de este tamaño, se generarán 64 valores enteros (número de características del dataset) que irán del 0 al 16. Esto reduce la dimensionalidad y da invarianza a pequeñas distorsiones.

Veamos un ejemplo:

0 0 0 11 9 0 0 0

Si transformamos la primera fila de matrices 4x4 en números enteros como acabamos de explicar, dará este resultado.

Estos 8 valores será los primeros que aparezcan en la fila correspondiente a este número, seguido de los 54 números que completan el mapa de bits.

Obviamente, seguirán el orden lógico de izquierda a derecha y de arriba abajo.

Como hemos dicho antes, el último número será el propio dígito manuscrito (en este caso se ve claramente que es el 6).

Los propios datos del repositorio ya vienen divididos en *training* y *test*. También te proporcionan los datos originales con los mapas de bits completos y sin normalizar, sin embargo, yo voy utilizar los que ya están procesados. Estos dos ficheros se llaman ‘optdigits.tra’ y ‘optdigits.tes’ y los utilizare en un directorio ‘datos’ localizado en el mismo sitio que el código.

Los datos de estos archivos son:

|  |  |  |
| --- | --- | --- |
| *Número de instancias:* |  |  |
| optdigits.tra | Training | 3823 |
| optdigits.tes | Testing | 1797 |

|  |  |  |
| --- | --- | --- |
| *Distribución de clases:* | | |
| Clase | **Nº ejemplos training** | **Nº ejemplos testing** |
| 0 | 376 | 178 |
| 1 | 389 | 182 |
| 2 | 380 | 177 |
| 3 | 389 | 183 |
| 4 | 387 | 181 |
| 5 | 376 | 182 |
| 6 | 377 | 181 |
| 7 | 387 | 179 |
| 8 | 380 | 174 |
| 9 | 382 | 184 |

## Airfoil Self-Noise Data Set

Este segundo conjunto de datos es más sencillo que el anterior. Las filas se componen sólo de 6 columnas, formadas por 5 características que determinan el nivel de presión sonora escalado, en decibelios. Este nivel será el valor de la última columna.

Los datos no están transformados ni modificados como en el ejemplo anterior, simplemente son los datos recogidos por la NASA para diferentes perfiles aerodinámicos NACA 0012. Estos perfiles comprenderán diferentes tamaños a varias velocidades en el túnel de viento y diferentes ángulos de ataque. La duración del perfil aerodinámico y la posición del observador fueron las mismas en todos los experimentos.

Veamos ahora a qué pertenece cada una de las 5 características mencionadas anteriormente, ordenadas por el número de columna al que pertenecen:

1. Frecuencia, en hercios.
2. Ángulo de ataque, en grados.
3. Longitud de la cuerda (trigonométricamente hablando), en metros.
4. Velocidad de flujo, en metros por segundo.
5. Grosor de desplazamiento lateral de aspiración, en metros.

Los datos vienen todos en un fichero ‘airfoil\_self\_noise.dat’ y, a diferencia del anterior sin dividir en *training* y *testing*. Este fichero también estará en el directorio ‘datos’ localizado en el mismo sitio que el código.

Los datos de estos archivos son:

|  |  |  |
| --- | --- | --- |
| *Número de instancias:* |  |  |
| airfoil\_self\_noise.dat | Training y testing | 1503 |

La distribución de clases, al ser tan extensa y tener tantos valores diferentes, no la podremos mostrar en una tabla como el anterior.

# 2. Preprocesado de los datos

Tanto en el primer conjunto de datos como en el segundo, realizaremos unas modificaciones para que el ajuste sea mejor. Estas modificaciones será las mismas en la mayoría de los modelos, sin embargo, se realizará pruebas cambiando parámetros y otras opciones.

Se utilizará una regularización ‘*Ridge* ’o ‘*l2* ’ para eliminar las características que no son relevantes del conjunto de datos. Es el método de regularización más utilizado para los problemas sin una solución única. Agrega una penalización equivalente al cuadrado de la magnitud de los coeficientes. A diferencia de la regularización ‘Lasso’ o ‘*l1*’, no reduce a cero algunos de los coeficientes, si no que reduce los coeficientes a un valor cercano de cero.

Esta regularización también reducirá a 0 las características que son constantes, lo que significa que estas características tienen el mismo valor para todas las muestras.

Otro método que utilizaré para transformar los datos será escalar las características y luego normalizarlas:

* Escalar. He utilizado la función ‘*MaxAbsScaler()*’, la cual escala cada característica por su valor máximo absoluto. Este estimador escala y traduce cada característica individualmente, de modo que el valor absoluto máximo de cada característica en el conjunto de entrenamiento será de 1. No desplaza o centra los datos, y por lo tanto no destruye ninguna dispersión.
* Normalizar. Simplemente aplicando la función ‘*normalize(X\_train)*’ escala las muestras individuales para tener una norma de unidad. Como ya veremos, esto es menos relevante después de realizar el escalado, no obstante, si no se realiza viene bastante bien sobre todo si planea usar una forma cuadrática como el producto puntual.

Veamos la implementación de estos dos cambios:

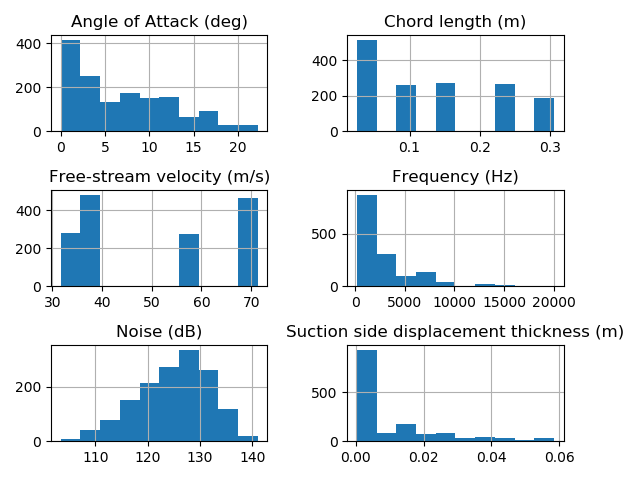
# Leemos a partir del fichero  
df = pd.read\_csv(**'datos/optdigits.tra'**)  
  
# Guardamos en una variable auxiliar los datos pero sin el  
# valor de las etiquetas  
df\_aux = df.copy()  
df\_aux = df\_aux.iloc[:, :-1]  
  
# Asignamos un escalador y lo aplicamos al conjunto de características  
scaled\_df = df\_aux.copy()  
scaler = MaxAbsScaler()  
scaled = scaler.fit\_transform(df\_aux)  
scaled\_df.loc[:,:] = scaled  
  
# Metemos en la variables el nuevo conjunto escalado  
trainX\_optdigits = np.array(scaled\_df)[:,:-1]  
trainY\_optdigits = np.array(df)[:,-1:]

# Normalizamos los datos del conjunto de entrenamiento

trainX\_optdigits = normalize(trainX\_optdigits)

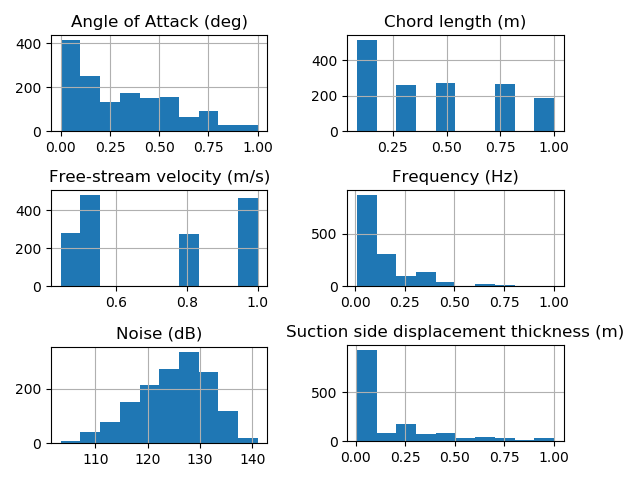
En este caso sólo lo estoy mostrando para el primer conjunto. Para el segundo también realizaremos lo mismo, lo único que cambiará será el nombre de las variables y del conjunto de datos.

Para el primer conjunto de datos, estas dos modificaciones van a ser útiles respecto a que tiene muchas características y muchas de ellas con valores constantes (o que varían muy poco) y se podrá reducir considerablemente el número de características a evaluar.

En el segundo conjunto de datos, lo de reducir características no va a ser tan relevante, puesto que cada dato sólo tiene 5. No obstante, escalar los datos va a ser muy importante por lo siguiente. Como hemos dicho antes, al existir un gran número de variables de clasificación no podíamos hacer una tabla, sin embargo, se mostrarán seis histogramas que representarán a cada una de las características del data set respectivamente.

Y su matriz de correlación:

Podemos ver que no existe ninguna relación directa entre parámetros que cambie mucho el resultado final, pero en los histogramas si hemos visto que para cada característica los valores son muy dispares. Por ejemplo, los valores de frecuencia van entre 0 y 20000 mientras que los valores del ángulo de ataque entre 0 y 25.

Cuando realizamos el escalado, obtenemos los siguientes valores:

Como vemos los dos histogramas son exactamente iguales, con la única diferencia de que ahora los valores están comprendidos entre 0 y 1, lo cual facilitará mucho la regresión posterior. A estos datos después se les realizará la regularización y el ajuste que hemos comentado antes, pero eso lo veremos cuando expliquemos cada modelo.

Mientras tanto, lo que vamos a hacer con los valores ya transformados va a ser dividirlos en los conjuntos de train y test.

# 3. Definición de los conjuntos training, validación y test

Para el primer conjunto de datos, como la división en training y test ya está realizada no vamos a utilizar ‘*cross-validation* ’, simplemente entrenaremos y validaremos con los datos de los ficheros proporcionados. En el segundo conjunto de datos, como sólo se nos proporciona un fichero con todos los datos juntos (es decir, hay que dividirlos igualmente) lo aprovecharemos para realizar ‘*cross-validation* ’ con todo el conjunto, no obstante, las pruebas y ejecuciones principales las realizaremos como hicimos en el primero.

Esta división la tendremos que realizar por nuestra cuenta. Para ello nos ayudaremos de una función de ‘*sklearn* ’ llamada ‘*train\_test\_*split ‘ y cuya implementación es la siguiente:

# Importamos el paquete

from sklearn.model\_selection import train\_test\_split

# Asignamos a estas variables la división en training y test

trainX\_airfoil, testX\_airfoil, trainY\_airfoil, testY\_airfoil = \  
 train\_test\_split(data[:,:data.shape[1]-1],  
 data[:,data.shape[1]-1],  
 test\_size=0.2)

Como podemos ver, esta función devuelve los dos conjuntos divididos y también separando en características y etiquetas. El parámetro que le pasamos, aparte de los parámetros de datos, es el tamaño que va a tener el test; y en mi caso lo he dejado con un 20% del conjunto total. El 80% restante pertenecerá será para realizar el entrenamiento.

# 4. Modelos lineales usados

A continuación, mostraremos los modelos lineales que se utilizarán tanto para clasificar el primer conjunto de datos como para hacer la regresión en el segundo. Por esta razón, estos modelos no serán los mismos para cada conjunto, sin embargo, he intentado utilizar los más parecidos posibles (por ejemplo, *‘SGDClassifier* ‘ y ‘*SGDRegressor* ‘ para el primero y el segundo respectivamente).

También he elegido los modelos teniendo en cuenta los que hemos visto en clase más en profundidad y los que hemos visto/implementado en clase de prácticas. Pese a esto, todos han sido sacados de ‘*sklearn* ‘, es decir, que no he utilizado mis propias implementaciones por optimizar un poco más el trabajo.

## Optical Recognition of Handwritten Digits Data Set

Los modelos utilizados, pese a ser diferentes, tienen varios parámetros en común, los cuales los he puesto todos iguales para que la comparación sea justa. Posteriormente, se cambiarán estos parámetros en los mejores modelos para ver cómo afecta. Estos parámetros son los siguientes:

* Penalty. Hace referencia a la regularización, y en este caso voy a usar la regularización ‘*Ridge* ’o ‘*l2* ’ para todos los casos.
* Max\_iter. Número máximo de iteraciones, por defecto puesto en 15000.
* Tol. Referente a la tolerancia, es decir, el umbral en el que se detendrá la ejecución. Por defecto puesto en 1e-4.

Pasemos a ver ahora los modelos lineales de clasificación utilizados.

### Perceptron