Aprendizaje Automático - Tercera Práctica
Dos caso de uso reales
Modelos Lineales

Sergio Quijano Rey - 72103503k
4º Doble Grado Ingeniería Informática y Matemáticas
sergioquijano@correo.ugr.es

4 de junio de 2021

Índice

Índice de figuras

Índice de cuadros

1. Problema de regresión

Los superconductores tienen la interesante propiedad de poder lograr resistencias al paso de la corriente muy cercanas a 0Ω. Sin embargo, esto solo ocurre cuando están por debajo de la temperatura crítica para este fenómeno, denotada como T_c.

Un superconductor con un un valor de T_c muy bajo no resultaría práctico en aplicaciones de ingeniería, pues para aprovechar sus propiedades interesantes debería realizarse un proceso de enfriamiento que potencialmente consumiría mucha energía. Por tanto, es interesante conocer los valores de T_c de los superconductores, para determinar si es viable o no su aplicación en distintos problemas.

No existe ningún modelo teórico para predecir el valor de T_c de nuevos superconductores, por tanto es interesante plantear un modelo de regresión de aprendizaje automático para predecir dicho valor de T_c [1].

1.1. Exploración del problema

1.1.1. Descripción del problema

Disponemos de dos archivos, train.csv y unique_m.csv. Este último archivo contiene las fórmulas químicas desglosadas de los superconductores con los que trabajamos. train.csv contiene 81 características de los superconductores, y el valor de T_c que queremos predecir.

En el propio paper [1] que se encuentra en la página del dataset con el que trabajamos, de UCI, se explica el tratamiento de los datos. En dicha sección, se detalla el proceso de extracción de las 81 características. A partir de las fórmulas codificadas en unique_m.csv, se extraen propiedades de los átomos que forman las moléculas. Al ser moléculas con más de un átomo, se toman estadísticos de las propiedades. Estas propiedades y estadísticos se detallan en *1.1.3. Descripción de las características.*

Por tanto, no parece factible que seamos capaces de obtener, a partir de conocimiento experto del problema, más *features*, de más alto nivel a ser posible, que resulten útiles para resolver el problema. Consecuentemente, no usaremos la información que nos pueda proporcionar unique_m.csv, ignorando este *dataset* por completo.

En el mismo paper, el autor comenta: *"We take an entirely data-driven approach"*. Por lo tanto, esto junto a nuestra falta de conocimiento sobre el problema, justifica que usemos ténicas estadísticas para establecer el conjunto de características a emplear (principalmente *PCA*), y una ténica como *cross-validation* para seleccionar el modelo a emplear, las transformaciones sobre los datos y distintos parámetros referentes al modelo escogido.

1.1.2. Problema a resolver

Queremos aprender una función objetivo de la forma:

$$f : \mathcal{X} \to \mathcal{Y}$$

donde $\mathcal{X}$ es el conjunto real de dimensión 81 (las 81 características de las que disponemos), e $\mathcal{Y}$ son valores reales, en el intervalo $[0, \infty]$. Como comentaremos en *1.1.3. Descripción de las características*, la unidad de medida de la temperatura son los Kelvin, y por tanto, tenemos una cota inferior de esta variable real.

Más adelante realizaremos transformaciones sobre el conjunto de datos original, por lo tanto, pasaremos de aprender un $f : \mathcal{X} \to \mathcal{Y}$ a aprender un $f : \hat{\mathcal{X}} \to \mathcal{Y}$, donde $\hat{\mathcal{X}}$ tendrá otra dimensión.

Por tanto quedan claros los elementos de un problema de regresión, queremos encontrar una función $g : \hat{\mathcal{X}} \to \mathcal{Y}$ de forma que $\forall x \in \hat{\mathcal{X}}, g(x) \approx f(x)$.

1.1.3. Descripción de las características

De nuevo, en el paper original [1] se describe el proceso de extracción de características, que pasamos a resumir brevemente.

Se parte de las siguientes propiedades de los átomos que componen las moléculas de los superconductores:

Variable	**Descripción**
Masa atómica	Masa total del protón y neutrón en reposo
Energía de primera ionización	Energía necesaria para eliminar una valencia del electrón
Radio Atómico	Radio atómico
Densidad	Densidad a una temperatura y presión estándar
Afinidad del electrón	Energía necesaria para añadir un electrón a un átomo neutro
Calor de fusión	Energía necesaria para pasar de estado sólido a líquido sin cambio de temperatura
Conductividad térmica	Coeficientes de conductividad térmica κ
Valencia	Número típico de enlaces químicos formados por el elemento

Cuadro 1: Propiedades de los elementos usadas para crear las *features*

Las *features* más importantes a la hora de predecir T_c son aquellas basadas en la *thermal conductivity*, *atomic radius*, *valence*, *electron affinity*, y *atomic mass* [1]. Con esto podríamos pensar en descartar el resto de *features* que no se basen en las anteriores, sin embargo, no tenemos el conocimiento suficiente sobre el problema para realizar este descarte con confianza, delegando esta decisión a la técnica *Principal Componente Analysis* que más adelante desarrollaremos.

A partir de esto, se calculan las siguientes *features*. Por cada material, en base a sus molécu-

las, se calculan los siguientes estadísticos por cada *feature* mostrada en la anterior tabla y por cada átomo de la molécula:

- Media
- Media ponderada
- Media geométrica
- Media geométrica ponderada
- Entropía
- Entropía ponderada
- Rango
- Rango ponderado
- Desviación estándar
- Desviación estándar ponderada

Algo importante a destacar es que la unidad de temperatura para T_c es *Kelvin*, por lo que esta variable estará acotada inferiormente por cero. La temperatura ambiente en kelvin está en torno a los $298K$, por lo tanto, valores cercanos a esta referencia serán los más interesantes a la hora de escoger o no un material como superconductor.

Las fórmulas para cada estadístico se pueden consultar en el ya mencionado *paper* [1]. Notar que tenemos siempre el estadístico y su versión ponderada.

Teniendo 8 variables, y 10 estadísticas por cada variable, llegamos a 80 características en el dataset. La característica que falta para llegar a las 81, es el número de elementos que compone la molécula del superconductor.

1.1.4. Exploración del *Dataset*

Antes de empezar a explorar los datos del problema, separamos el conjunto de *training* y de *test*. No queremos saber nada sobre el *test_dataset* durante esta exploración de los datos, para evitar caer en el *data snooping*. Esta separación de los datos la realizamos con la función split_data en la que hacemos:

```
df_test, df_train = train_test_split(df, test_size =
    test_percentage, shuffle = True, stratify = None)
```

Notar que estamos mezclando los datos pues shuffle = True, con ello, y teniendo en cuenta que disponemos de muchos datos, queremos tener una muestra de entrenamiento representativa de los datos. Por ejemplo, no queremos que tengamos desbalanceos en la variable de salida, es decir, que en *train* tengamos filas con valores de T_c bajo, mientras que en *test* tengamos filas con valores de T_c altos, o viceversa. Como no tenemos clases, y al tener una cantidad tan grande de datos, no hacemos *stratify != None*. Confiamos en que la mezcla aleatoria haga que nuestras muestras sean representativas y balanceadas, en el sentido que ya se ha especificado.

Partimos de un *dataset* con 21263 ejemplos. Al separar en *train* y *test*, nos quedamos con 17010 y 4253 ejemplos, respectivamente.

Con la función explore_training_set hacemos una pequeña exploración estadística de los datos, en la que mostramos una tabla con las estadísticas de las columnas. Dicha tabla con el análisis descriptivo de los atributos del conjunto de entrenamiento se muestra en *Tabla 2* (tabla que se presenta en dos partes, debido a la gran extensión):

name	mean	median	var	std	min	max	p25	p75
number_of_elements	4.11	4.00	2.08e+0	1.44	1.00	9.00	3.00	5.00
mean_atomic_mass	87.42	84.78	8.80e+2	29.67	6.94	208.98	72.38	100.35
wtd_mean_atomic_mass	72.95	60.84	1.12e+3	33.56	6.42	208.98	52.07	86.07
gmean_atomic_mass	71.17	66.36	9.62e+2	31.02	5.32	208.98	57.78	78.11
wtd_gmean_atomic_mass	58.54	39.93	1.34e+3	36.69	1.96	208.98	35.18	73.05
entropy_atomic_mass	1.16	1.19	1.34e-1	0.36	0.00	1.98	0.96	1.44
wtd_entropy_atomic_mass	1.06	1.14	1.62e-1	0.40	0.00	1.95	0.76	1.35
range_atomic_mass	115.39	122.90	2.98e+3	54.64	0.00	207.97	78.09	153.96
wtd_range_atomic_mass	33.20	26.52	7.33e+2	27.07	0.00	205.58	16.73	38.33
std_atomic_mass	44.31	45.02	4.02e+2	20.05	0.00	101.01	32.89	58.97
wtd_std_atomic_mass	41.33	44.27	3.99e+2	19.98	0.00	101.01	28.53	53.58
mean_fie	770.51	765.75	7.76e+3	88.11	502.50	1313.10	723.74	797.15
wtd_mean_fie	870.52	889.69	2.04e+4	142.98	502.50	1348.02	739.28	1003.97
gmean_fie	738.37	728.82	6.23e+3	78.95	502.50	1313.10	692.54	766.46
wtd_gmean_fie	832.96	855.51	1.43e+4	119.63	502.50	1327.59	720.64	937.55
entropy_fie	1.29	1.35	1.46e-1	0.38	0.00	2.15	1.08	1.55
wtd_entropy_fie	0.92	0.91	1.12e-1	0.33	0.00	2.03	0.75	1.06
range_fie	572.06	764.10	9.62e+4	310.24	0.00	1304.50	259.10	810.60
wtd_range_fie	482.65	508.21	5.03e+4	224.47	0.00	1251.85	290.90	690.55
std_fie	215.56	266.29	1.21e+4	110.16	0.00	499.67	113.56	297.52
wtd_std_fie	223.66	258.10	1.63e+4	127.88	0.00	477.81	92.64	342.60
mean_atomic_radius	157.85	160.25	4.09e+2	20.24	48.00	253.00	149.00	169.80
wtd_mean_atomic_radius	134.77	126.02	8.30e+2	28.81	48.00	253.00	112.13	158.38
gmean_atomic_radius	144.33	142.80	4.91e+2	22.16	48.00	253.00	133.54	155.93
wtd_gmean_atomic_radius	121.07	113.27	1.28e+3	35.82	48.00	253.00	89.22	151.06
entropy_atomic_radius	1.26	1.32	1.41e-1	0.37	0.00	2.14	1.06	1.51
wtd_entropy_atomic_radius	1.12	1.24	1.66e-1	0.40	0.00	1.90	0.84	1.42
range_atomic_radius	139.13	171.00	4.53e+3	67.34	0.00	256.00	80.00	205.00
wtd_range_atomic_radius	51.41	43.04	1.23e+3	35.12	0.00	240.16	28.53	60.57
std_atomic_radius	51.54	58.66	5.25e+2	22.92	0.00	115.50	35.00	69.42
wtd_std_atomic_radius	52.26	59.74	6.40e+2	25.31	0.00	97.14	31.82	73.66
mean_Density	6115.33	5329.08	8.16e+6	2858.00	1.42	22590.00	4506.75	6769.93
wtd_mean_Density	5278.72	4386.11	1.05e+7	3240.74	1.42	22590.00	2998.57	6422.80
gmean_Density	3464.31	1339.97	1.37e+7	3711.86	1.42	22590.00	883.11	5802.35
wtd_gmean_Density	3126.70	1525.86	1.59e+7	3991.46	0.68	22590.00	66.76	5763.29
entropy_Density	1.07	1.09	1.18e-1	0.34	0.00	1.95	0.90	1.32
wtd_entropy_Density	0.85	0.88	1.03e-1	0.32	0.00	1.70	0.68	1.07
range_Density	8672.52	8958.57	1.69e+7	4118.33	0.00	22588.57	6648.00	9778.57
wtd_range_Density	2914.45	2082.95	5.86e+6	2421.24	0.00	22434.16	1659.70	3427.42
std_Density	3419.54	3294.07	2.83e+6	1682.52	0.00	10724.37	2819.49	4004.27
wtd_std_Density	3318.18	3623.83	2.61e+6	1617.33	0.00	10410.93	2564.34	3956.79
mean_ElectronAffinity	77.04	73.10	7.76e+2	27.86	1.50	326.10	62.09	85.85
wtd_mean_ElectronAffinity	92.77	102.73	1.04e+3	32.35	1.50	326.10	73.39	110.73
gmean_ElectronAffinity	54.49	51.53	8.47e+2	29.11	1.50	326.10	33.70	67.57
wtd_gmean_ElectronAffinity	72.42	73.08	1.00e+3	31.70	1.50	326.10	50.87	89.96
entropy_ElectronAffinity	1.06	1.13	1.18e-1	0.34	0.00	1.76	0.87	1.34
wtd_entropy_ElectronAffinity	0.77	0.78	8.25e-2	0.28	0.00	1.67	0.65	0.87
range_ElectronAffinity	121.03	127.05	3.50e+3	59.19	0.00	349.00	86.10	138.63
wtd_range_ElectronAffinity	59.32	71.12	8.29e+2	28.80	0.00	218.69	33.99	76.70

Cuadro 2: Exploración estadística de los atributos del conjunto de entrenamiento, parte 1

name	mean	median	var	std	min	max	p25	p75
std_ElectronAffinity	49.01	51.12	4.81e+2	21.94	0.00	162.89	38.43	56.52
wtd_std_ElectronAffinity	44.50	48.16	4.23e+2	20.58	0.00	169.07	33.34	53.43
mean_FusionHeat	14.32	9.33	1.28e+2	11.31	0.22	105.00	7.58	17.22
wtd_mean_FusionHeat	13.89	8.41	2.04e+2	14.30	0.22	105.00	5.05	18.54
gmean_FusionHeat	10.13	5.27	1.01e+2	10.07	0.22	105.00	4.11	13.59
wtd_gmean_FusionHeat	10.16	4.96	1.72e+2	13.14	0.22	105.00	1.32	16.42
entropy_FusionHeat	1.09	1.11	1.42e-1	0.37	0.00	2.03	0.82	1.37
wtd_entropy_FusionHeat	0.91	0.99	1.38e-1	0.37	0.00	1.74	0.66	1.15
range_FusionHeat	21.21	12.87	4.19e+2	20.47	0.00	104.77	12.87	23.54
wtd_range_FusionHeat	8.25	3.45	1.31e+2	11.45	0.00	102.38	2.34	10.49
std_FusionHeat	8.35	4.94	7.60e+1	8.72	0.00	51.63	4.26	9.10
wtd_std_FusionHeat	7.74	5.51	5.36e+1	7.32	0.00	51.68	4.60	8.02
mean_ThermalConductivity	89.48	96.17	1.48e+3	38.57	0.02	332.50	60.50	111.00
wtd_mean_ThermalConductivity	81.57	73.55	2.10e+3	45.87	0.02	406.96	53.77	99.04
gmean_ThermalConductivity	29.80	14.28	1.16e+3	34.08	0.02	317.88	8.33	41.73
wtd_gmean_ThermalConductivity	27.32	6.11	1.62e+3	40.32	0.02	376.03	1.08	47.07
entropy_ThermalConductivity	0.72	0.73	1.05e-1	0.32	0.00	1.63	0.45	0.95
wtd_entropy_ThermalConductivity	0.53	0.54	1.00e-1	0.31	0.00	1.61	0.24	0.77
range_ThermalConductivity	250.06	399.48	2.52e+4	158.79	0.00	429.97	86.00	399.97
wtd_range_ThermalConductivity	62.11	56.47	1.89e+3	43.56	0.00	401.44	29.25	91.93
std_ThermalConductivity	98.60	134.63	3.61e+3	60.15	0.00	214.98	37.55	153.51
wtd_std_ThermalConductivity	95.98	113.36	4.07e+3	63.81	0.00	213.30	31.89	162.66
mean_Valence	3.20	2.83	1.09e+0	1.04	1.00	7.00	2.33	4.00
wtd_mean_Valence	3.16	2.63	1.42e+0	1.19	1.00	7.00	2.11	4.05
gmean_Valence	3.06	2.61	1.10e+0	1.04	1.00	7.00	2.28	3.77
wtd_gmean_Valence	3.06	2.43	1.39e+0	1.17	1.00	7.00	2.09	3.94
entropy_Valence	1.29	1.36	1.55e-1	0.39	0.00	2.14	1.06	1.58
wtd_entropy_Valence	1.05	1.16	1.45e-1	0.38	0.00	1.94	0.76	1.33
range_Valence	2.04	2.00	1.55e+0	1.24	0.00	6.00	1.00	3.00
wtd_range_Valence	1.48	1.06	9.68e-1	0.98	0.00	6.99	0.91	1.92
std_Valence	0.84	0.80	2.37e-1	0.48	0.00	3.00	0.47	1.21
wtd_std_Valence	0.67	0.50	2.09e-1	0.45	0.00	3.00	0.30	1.02
critical_temp	34.37	20.00	1.17e+3	34.25	0.00	185.00	5.30	63.00

Cuadro 2: Exploración estadística de los atributos del conjunto de entrenamiento, parte 2

No mostramos el valor *missing values*, porque en todos los casos son cero, así que no tenemos que preocuparnos de cómo afrontar este problema. Tampoco mostramos el valor de *type*. Todos los valores son *float64*, salvo *number_of_elements*, *range_atomic_radius* y *range_Valence*, que son *int64*

La tabla deja claro que los rangos de las variables son muy dispares, así como las desviaciones típicas. Por ejemplo, wtd_mean_ThermalConductivity toma un rango de valores que va desde 0 hasta 406.96, mientas que por ejemplo entropy_ThermalConductivity va desde 0 hasta 1.63. Lo mismo se puede decir de las desviaciones típicas. Muchos algoritmos y modelos son sensibles a rangos de valores dispares entre distintas características. Otros directamente esperan que se siga una distribución parecida a una normal para tener un comportamiento decente. Por tanto, y como no es perjudicial realizar una estandarización, queda justificada la posterior estandarización que vamos a llevar a cabo.

Otro elemento a tener en cuenta es que todas las variables son reales o enteras, y por tanto, tampoco es necesaria una técnica como *one hot encoding* para codificar variables categóricas.

Con estos datos, podemos mostrar *boxplots* de las variables, pero tampoco extraeríamos demasiada información, pues más tarde vamos a estandarizar los datos, como ya hemos comen-

tado, y además, vamos a eliminar los *outliers*.Sin embargo, la variable de salida *critical_temp* no va a ser estandarizada ni eliminados los *outliers* asociados a esta columna. Mostramos su *boxplot*:

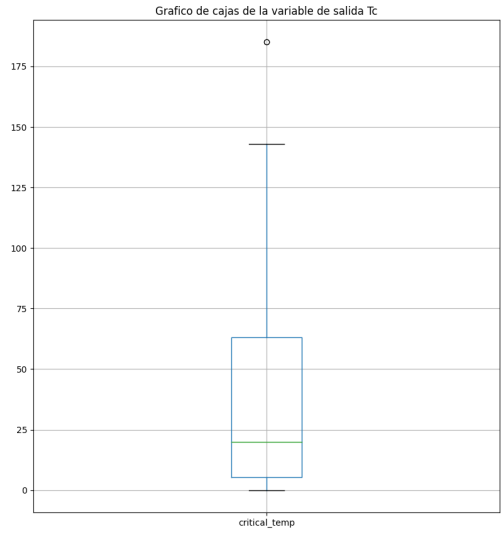

Figura 1: Boxplot de la temperatura crítica

La caja del gráfico muestran los extremos de los extremos que fijan el percentil 25 y 75. Por tanto, podemos ver que nuestros datos de entrada están muy acumulados en valores de salida bajos. Es decir, la mayoría de datos con los que trabajamos están asociados a superconductores con un T_c bajo, y por tanto menos interesantes. Esto mismo se comenta en el paper original, en la figura 4 [1].

Por tanto, debemos preservar estos *outliers* en la variable de salida, pues son precisamente los datos que nos interesa predecir. Podríamos intentar solucionar este desbalanceo eliminando datos en la parte de mayor acumulación (datos de baja temperatura T_c). Sin embargo, a vista de lo desplazada que está la caja del gráfico, eliminaríamos demasiados datos, con lo que seguramente no mejoraríamos el rendimiento de la función aprendida. Además, nuestro objetivo no es aprender bien la función para superconductores con T_c alto (aunque sean los más interesantes), sino aprender bien la función f que ya hemos descrito anteriormente.

1.2. Preprocesado de los datos

1.2.1. Eliminación de outliers

Antes de realizar normalización, debemos eliminar los *outliers*. Estos son aquellos valores que están a una distancia de la media de más de 3 veces la desviación típica. Tenemos distintas características por cada dato, así que definimos los *outliers* como aquellas filas que, en alguna de las variables que definen las columnas, se desvían como ya hemos especificado. Podríamos haber optado por técnicas que detectasen *outliers* basándonos en más de una variable, pero por simplicidad, seguimos el procedimiento ya indicado. La opción multivariable preserva más datos, pero tenemos un *dataset* lo suficientemente grande como para permitir el borrar más filas.

En nuestro caso, por ser menos restrictivos, establecemos el límite en 4 veces la desviación típica.

Todo esto está fundamentado en que, en una distribución normal, el 99,74 % de los datos se encuentran en el intervalo $[\mu - \sigma, \mu + \sigma]$. Al tener una gran cantidad de datos, por el teorema central del límite, podemos suponer que nuestra distribución de datos se aproxima a una distribución normal (multivariante al tener varias variables).

Además, hacemos esto antes de estandarizar, pues para estandarizar, usamos los estadísticos media y desviación típica, que son muy sensibles a los valores *outliers* [2]. También afectan los *outliers* al procedimiento de *PCA*, pues se basa en estadísticos altamente afectados por dichos *outliers* [3].

Previamente hemos justificado que no vamos a borrar *outliers* respecto a la variable de salida.

El código que borra los outliers se encuentra en la función remove_outliers [1]. Usamos una orden de acceso de la librería pandas en la que usamos el estadístico zscore, de la librería scipy. Este valor zscore lo que mide es el número de desviaciones típicas en las que un punto dista de la media de la variable aleatoria, es decir:

$$z_{score}(x) := \frac{x - \mu}{\sigma}$$

Borramos aquellos valores que, en alguna variable aleatoria columna, tengan un valor absoluto de zscore mayor o igual que 4.

Tras ejecutar el eliminado de *outliers*, eliminamos el 8.78 % de los datos. Teniendo en cuenta la gran cantidad de datos de los que disponemos, junto al hecho de que en [1] comentan los autores que se quedan con el 67 % de los datos originales, para acabar con un *dataset* de calidad, queda justificada esta pérdida de datos en pro de un conjunto de datos más limpio y de calidad.

En este punto nos preocupamos de haber eliminado, sin fijarnos en la variable de salida, filas con valores de salida interesantes o que desbalanceen aún más el conjunto de datos. Sin embargo, computamos unas cuantas estadísticas de las filas eliminadas, respecto de la columna de salida T_c. La media de los datos eliminados es 12.82, y la desviación típica es 22.34. Por

[1] En la sección *4. Consideraciones adicionales* indicamos la librería adicional que hemos empleado para el cálculo del *z-score*

tanto estamos eliminando mayoritariamente filas con T_c bajos, así que no vemos que estemos introduciendo aún más desbalanceo.

1.2.2. Estandarización

En este proceso, buscamos que las variables aleatorias de nuestro conjunto de datos queden con media cero y desviación típica uno. Este proceso no hace que las variables estén en un rango de valores similares, sin embargo, en este problema de regresión no parece ser importante. No usamos técnicas basadas en proximidad como *nearest neighbour* o *SVM*, en las que una normalización sería más adecuada la normalización [4]. En normalización conseguiríamos que todas las variables estuviesen en el rango $[0, 1]$.

En estandarización, aplicamos la siguiente operación a todas las variables aleatorias que conforman nuestro conjunto de datos:

$\mathbb{X}' = \frac{\mathbb{X}-\mu}{\sigma}$

El código usado para estandarizar se encuentra en standarize_dataset. De nuevo, recibe como parámetro el conjunto de test. Este conjunto no se usa para calcular la transformación que representa la estandarización. Este cálculo solo toma información de los datos de entrenamiento. Por tanto, sobre el conjunto de test, solo se aplica la misma transformación.

En dicho código usamos la clase StandarScaler de sklearn, que hace justo lo que hemos especificado [5].

Aplicamos estandarización tanto al conjunto original de datos, al que solo hemos borrado outliers, y tras aplicar la técnica *PCA*, aplicaremos de nuevo estandarización a los datos transformados. Notar que es importante la estadarización previamente a *PCA*, pues como se comentará más tarde, rangos de la desviación típica dispares puede hacer que una variable domine sobre las demás desproporcionadamente, obteniendo resultados no deseados.

En dicho código se puede observar que no estamos estandarizando la variable de salida, pues esto no tiene sentido. En datos no vistos en el entrenamiento, nos llegan variables de entrada pero no de salida. Así que la predicción debe hacerse en el conjunto aleatorio que representa la variable aleatoria de salida original.

Mostramos algunos de los datos, no todos pues no tiene mayor interés, del conjunto de datos original tras estandarizar:

name	**mean**	**median**	**var**	**sdt**	**min**	**max**	**p25**	**p75**
numberof_elements	1.95e-16	-0.07	1.00	1.00	-2.15	3.38	-0.77	0.61
meanatomic_mass	-5.66e-17	-0.08	1.00	1.00	-2.71	4.09	-0.50	0.43
wtdmean_atomic_mass	3.29e-17	-0.36	1.00	1.00	-1.98	4.05	-0.62	0.39
gmeanatomic_mass	-2.81e-16	-0.15	1.00	1.00	-2.12	4.44	-0.43	0.22
...	...	...	...	...	...	...	...	...
wtdrange_Valence	-4.75e-17	-0.42	1.00	1.00	-1.50	5.59	-0.57	0.44
stdValence	-3.22e-16	-0.08	1.00	1.00	-1.72	4.42	-0.76	0.75
wtdstd_Valence	7.12e-17	-0.38	1.00	1.00	-1.47	5.08	-0.80	0.75

Cuadro 3: Conjunto de datos sin aplicar *PCA* tras la *estandarización*

En la tablas vemos que acabamos con desviación típica 1 y media prácticamente cero (notar que tenemos valores en órdenes de magnitud de 10^{-16} o 10^{-17}, es decir, prácticamente cero). Aunque esta técnica de escalado no se centre en normalizar el rango de valores, si que hace que los rangos se normalicen algo. Por ejemplo, en la *Tabla 3. Conjunto de datos sin aplicar* PCA *tras la* estandarización, el valor de *meanatomic_mass* se mueve ahora en un rango $[-2{,}71, 4{,}09]$, mientras que en la *Tabla 2. Exploración estadística de los atributos del conjunto de entrenamiento, parte 2* podemos ver que se movía en un rango $[6{,}94, 208{,}98]$. Así que aunque no estemos explícitamente preocupándonos por el rango de las variables, sí que estamos haciendo que no sean rangos tan amplios ni rangos tan dispares entre distintas variables.

1.2.3. Principal Component Analysis

Con esta técnica buscamos reducir la dimensionalidad de nuestro conjunto de datos, manteniendo el máximo de la variabilidad original (que es lo que nos permite llevar a cabo un proceso de aprendizaje).

Buscamos una base ortonormal de un espacio $\mathbb{R}^{\hat{d}}$ donde $\hat{d} << d$, es decir, el nuevo espacio euclídeo tiene una dimensión mucho menor que el espacio original. Todo esto gracias a calcular los valores propios de la matriz de covarianzas del conjunto de datos original [6] [7]. Con ello, y usando propiedades de vectores y espacios propios, expresamos en espacio con vectores que están linealmente incorrelados.

Además, esta técnica devuelve los vectores de la base ordenados según la varianza que explican del conjunto de datos original.

En nuestro caso, esta transformación del espacio la realizamos gracias a la función apply_PCA . Podemos especificar el número de variables con el que nos queremos quedar, como el porcentaje de varianza que queremos alcanzar. Notar que pasamos como parámetro el conjunto de test. Este conjunto de test **no se usa para calcular la transformación**, solo se pasa para aplicar la misma transformación calculada, de nuevo, exclusivamente usando los datos del conjunto de entrenamiento.

Un detalle destacable es que antes de aplicar *PCA* debemos estandarizar el conjunto de datos, pues de otra forma trabajaríamos con peores resultados. De otra forma, rangos dispares en las desviaciones permitiría que una variable dominase demasiado a las otras variables al tener las estadísticas en rangos dispares [8].

Cuando buscamos un 99 % de varianza explicada, obtenemos una transformación del espacio $\mathbb{R}^{81}$ al espacio $\mathbb{R}^{30}$, es decir, hemos acabado con el 37,04 % de las variables originales, manteniendo el 99 % de la variableidad del dataset original. Con esto seremos capaces de realizar transformaciones no lineales de los datos, buscando un mejor ajuste lineal a la muestra de entrenamiento. Una reducción tan drástica tiene sentido, pues tenemos 8 variables dependientes de los átomos, que ya de por sí estarán en cierto grado correladas, y 10 estadísticas moleculares, que podemos suponer altamente correladas.

Las estadísticas del conjunto de datos tras la transformación se refleja en la siguiente tabla:

Columna	mean	median	var	sdt	min	max	p25	p75
0	-1.35e-16	-3.05	31.45	5.60	-8.69	18.00	-4.69	4.44
1	1.23e-17	-0.29	8.54	2.92	-7.42	17.15	-1.69	1.10
2	1.53e-16	0.26	7.66	2.76	-10.17	10.29	-1.78	2.17
3	1.15e-17	0.09	6.42	2.53	-8.03	11.75	-1.48	1.36
4	4.50e-17	0.23	4.79	2.18	-8.72	11.06	-1.00	1.01
5	3.43e-17	-0.28	3.05	1.74	-7.73	10.46	-1.15	0.91
6	3.60e-18	-0.11	2.95	1.71	-7.06	9.57	-0.82	0.78
7	-5.65e-17	-0.05	2.56	1.60	-7.79	9.03	-0.65	0.78
8	-8.47e-18	-0.10	1.90	1.38	-6.53	8.11	-0.69	0.73
9	-7.70e-18	-0.08	1.60	1.26	-6.10	8.13	-0.81	0.76
10	8.10e-19	0.04	1.46	1.21	-6.82	7.16	-0.57	0.64
11	-2.40e-17	-0.04	1.17	1.08	-3.88	6.32	-0.46	0.50
12	2.50e-17	0.04	0.94	0.97	-4.42	4.84	-0.60	0.52
13	8.87e-18	-0.05	0.81	0.90	-4.47	4.75	-0.42	0.48
14	3.25e-17	-0.02	0.78	0.88	-4.08	6.73	-0.36	0.38
15	-5.81e-17	-0.02	0.63	0.79	-4.76	4.84	-0.40	0.42
16	-2.21e-17	-0.09	0.58	0.76	-3.44	5.50	-0.39	0.27
17	1.80e-17	-0.01	0.44	0.66	-2.50	4.05	-0.37	0.38
18	1.38e-17	-0.01	0.39	0.62	-3.76	4.72	-0.31	0.27
19	-1.69e-17	-0.03	0.30	0.55	-2.42	3.17	-0.32	0.29
20	1.81e-17	-0.03	0.24	0.49	-1.75	2.88	-0.34	0.29
21	-2.99e-17	-0.00	0.22	0.47	-2.56	3.53	-0.21	0.21
22	1.13e-17	-0.00	0.20	0.45	-2.59	4.32	-0.26	0.28
23	4.11e-18	-0.00	0.16	0.40	-2.22	3.08	-0.22	0.24
24	-3.87e-18	-0.00	0.15	0.39	-2.20	2.94	-0.23	0.22
25	-4.30e-17	-0.00	0.14	0.38	-2.09	2.79	-0.21	0.21
26	-2.97e-17	-0.02	0.13	0.37	-2.01	2.37	-0.23	0.21
27	-2.85e-18	0.00	0.11	0.34	-1.67	1.92	-0.17	0.17
28	-1.64e-17	0.00	0.10	0.32	-2.03	2.64	-0.16	0.15
29	3.84e-18	0.01	0.09	0.30	-1.60	2.05	-0.17	0.16
30	-3.34e-17	-0.02	0.08	0.28	-1.47	1.92	-0.15	0.15

Cuadro 4: Estadísticas de las *features* tras aplicar *PCA*

Es claro que no controlamos la transformación, por tanto no tenemos interpretación de lo que representa cada columna. También es claro que tenemos desviaciones típicas en órdenes de magnitud distintas y rangos (mínimo y máximo) completamente dispares. Por tanto, tanto como con el conjunto de datos original como en el conjunto tras aplicar *PCA*, es necesario realizar una estandarización o normalización de los datos.

Estandarizamos estos nuevos datos, pues es claro que los rangos y las desviaciones son bastante dispares. Mostramos algunos de los datos, no tiene mayor interés mostrar las 30 filas:

name	mean	median	var	sdt	min	max	p25	p75
0	-1.16e-17	-0.54	1.00	1.00	-1.55	3.21	-0.83	0.79
1	1.87e-18	-0.10	1.00	1.00	-2.54	5.87	-0.57	0.37
2	-1.42e-17	0.09	1.00	1.00	-3.67	3.71	-0.64	0.78
...	...	...	...	...	...	...	...	...
28	-1.48e-17	0.01	1.00	1.00	-6.33	8.21	-0.50	0.46
29	1.73e-17	0.05	1.00	1.00	-5.23	6.70	-0.57	0.53
30	1.00e-17	-0.06	1.00	1.00	-5.13	6.69	-0.53	0.54

Cuadro 5: Conjunto de datos tras aplicar *PCA* y *estandarización*

Y de nuevo, vemos que tenemos media aproximadamente cero y desviación típica igual a uno, como buscábamos

1.3. Selección del modelo

En esta sección vamos a emplear la técnicas de *Cross Validation* para la selección del modelo y los parámetros empleados durante el aprendizaje. Vamos a emplear dos veces CV como vamos a detallar más adelante.

En realidad estamos usando la técnica de *K-Fold Cross Validation*, con $K = 10$. En esta técnica, tomamos nuestro *dataset* de entrenamiento y lo dividimos en K *folds* o subgrupos. Una vez hecho esto, realizamos K veces el proceso de tomar un *fold* como conjunto de validación, y los restantes $K - 1$ *folds* para entrenar el modelo candidato. Sobre el *fold* de validación calculamos una métrica de error. Así tenemos K entrenamientos con K métricas, de las que podemos calcular estadísticas como media, mínimo y máximo, ...

En las funciones show_cross_validation_step1 y show_cross_validation_step2 aplicamos dos veces el proceso de *Cross Validation*.

1.3.1. Selección de la métrica de error

En ambas fases de *Cross Validation*, usaremos la misma métrica de error. En este caso, usamos el **Error cuadrático Medio**. Es una de las pocas métricas de error que conocemos para los problemas de regresión. De todas formas, elegimos esta métrica porque es fácilmente interpretable, el término cuadrático castiga más los valores que se predicen peor que otra métrica como el error absoluto medio. El *ECM* es una métrica de error que se usa en el proceso de aprendizaje, directamente en el método de los mínimos cuadrados ordinario ordinario, o como término del error aumentado, donde el *ECM* es el sumando de la métrica del error que va acompañada de un término de penalización de la parte de regularización.

A la hora de especificar la métrica de error con sklearn, usamos el error cuadrático medio negativo. Esto porque sklearn busca maximizar esta *score* en otras funciones. Así, un problema de minimizar el error, pasa a ser un problema de maximización cambiando el signo de la función a optimizar [9].

1.3.2. Primera etapa - Modelos candidatos

A la hora de resolver este problema tenemos como modelo el ajuste de un hiperplano a los datos. Es decir, nuestra clase de funciones que representa el modelo viene dada por:

$$\mathcal{H} := \{f_w / w \in \mathbb{R}^D\}$$

donde $w \in \mathbb{R}^d$ son los parámetros que definen cada una de las hipótesis pertenecientes a la clase de funciones, que especifican la función de la siguiente forma:

$$f_w(x) := w^T x, \forall x \in \mathbb{R}^D$$

Así que lo buscamos elegir en esta fase de *Cross Validation* es:

- El conjunto de datos y la transformación que queremos aplicar sobre los datos: datos a los que no aplicamos PCA y sin transformaciones, datos a los que aplicamos PCA y aplicamos transformaciones polinómicas
- El algoritmo de aprendizaje: mínimos cuadrados ordinarios, Ridge o Lasso

Respecto a las transformaciones de los datos, a falta de conocimiento experto sobre el problema para guiar las transformaciones empleadas, probamos con transformaciones polinómicas ϕ_q. Fijado el orden q, calculamos todos los polinomios en varias variables de hasta orden q. Por ejemplo, con $q = 3$, podemos encontrar en el vector transformado los elementos $x, x^2, xy, x^2y, y^2x, xyz, \ldots$. En código esto lo conseguimos con la función de sklearn PolynomialFeatures [10].

El primer modelo consiste en minimizar el error cuadrático medio a través de la matriz pseudoinversa [11]. Por tanto, no tenemos que preocuparnos de parámetros de procesos iterativos como la tolerancia.

En segundo modelo consiste en minimizar el error aumentado en el que el término de penalización de regularización viene dado por $\lambda||w||_2^2$. Con esto, favorecemos soluciones con valores en los pesos no muy grandes, aunque no necesariamente cero. Se realiza un proceso iterativo, donde los valores por defecto son [12]:

- Máximo de iteraciones: 1000 iteraciones por defecto. A vista de nuestros datos, parece un número suficiente de iteraciones
- Tolerancia: por defecto, 10^{-3}, diferencia de error mínima entre dos iteraciones. Teniendo en cuenta de que llegaremos a errores en el intervalo $[300, 1800]$, parece una tolerancia mucho más que aceptable
- Alpha: valor que nosotros hemos llamado en el curso λ. Término de penalización para la regularización. Lo establecemos nosotros a un valor de 0.05. Parece sensato pues en otros muchos problemas hemos visto que se emplea valores en el intervalo $[10^{-2}, 1]$. Además tenemos una dimensionalidad pequeña así que no parece necesario tomar un valor alto para λ

En el caso de Lasso, tenemos la misma situación que con Ridge pero tomando el error para la parte de regularización como $\sum_{k=1}^{d} |w_k|$. En este caso, estamos favoreciendo el que muchos parámetros sean cero. Por tanto, es un buen modelo para selección de características. Usamos los mismos parámetros que especificados para Ridge.

Respecto al paso al código de los modelos ya detallados, podemos encontrar toda la información de sklearn en la documentación oficial [13].

Estamos considerando dos regularizadores distintos (los que se usan en Lasso y Ridge) en vez de decantarnos directamente por uno de los dos en base a alguna intuición que tengamos sobre los datos. Si tuviésemos que elegir solo uno, elegiríamos *Ridge* porque no estamos haciendo selección de características ni con una gran cantidad de variables. Sin embargo, estamos haciendo las transformaciones polinómicas sin tener mucha idea de si muchos de los monomios obtenidos van a ser útiles o no. Por eso confiamos en que Lasso no de peso a muchos monomios irrelevantes, dándole peso a algunos monomios relevantes de forma automática. Sin embargo, si esto falla, confiamos más en el empleo de *Ridge*.

Por otro lado, como hemos sido capaces de emplear la pseudoinversa sin problemas de tiempos de cómputo, empleamos esta técnica *one step solution* en vez de un enfoque iterativo que podríamos haber empleado con *SGDRegressor*. Los momentos en los que el ordenador no es capaz de realizar los cómputos, es porque la transformación polinómica es demasiado grande.

1.3.3. Resultados de *Cross-Validation*, primera etapa

Con la orden cv = KFold(n_splits=10, shuffle=True) especificamos los 10 *folds* y que se use mezclado aleatorio para elegir los elementos de los *folds*.

En el conjunto de datos al que aplicamos *PCA*, probamos las transformaciones polinómicas en el conjunto $\{1, 2\}$. No aplicamos transformaciones de orden superior pues nuestro ordenador no es capaz de calcular estas transformaciones. Al conjunto de datos al que no aplicamos *PCA*, no aplicamos transformaciones polinómicas, pues de nuevo nuestro ordenador no es capaz realizar los cómputos

Al usar *Lasso*, el código lanza algunos errores porque no alcanza solución estable en el número máximo de iteraciones dado. Sin embargo, el código automáticamente vuelve a intentarlo con otra solución inicial aleatoria, llegando a encontrar la solución en el número máximo de iteraciones especificado.

Los resultados de *Cross Validation* se resumen en la siguiente tabla:

Modelo	PCA No PCA	Orden de la transformación polinómica	Valor medio	Valor mínimo	Valor máximo
Lineal	PCA	1	-365.39	-411.91	-347.46
Lineal	PCA	2	-229.56	-254.66	-210.64
Ridge	PCA	1	-365.59	-397.05	-345.15
Ridge	PCA	2	-231.89	-259.39	-200.73
Lasso	PCA	1	-365.45	-402.61	-343.86
Lasso	PCA	2	-231.36	-267.28	-213.01
Lineal	No PCA	1	-310.50	-322.55	-282.40
Ridge	No PCA	1	-310.77	-335.69	-292.88
Lasso	No PCA	1	-328.42	-355.69	-310.94

Cuadro 6: Resultados de *Cross Validation*, primera fase

La tabla deja claro que lo mejor es aplicar *PCA* y transformaciones polinómicas de segundo grado. Sería interesante conocer los resultados de aplicar transformaciones de hasta orden 3 en una máquina más potente, con mas memoria para almacenar los cálculos. El claro ganador es tomar *PCA*, transformación de orden 2 y modelo lineal normal. Sin embargo, el segundo mejor, *PCA* con orden 2 y Lasso, difiere en error cuadrático medio por tan solo un valor de 1,8. Como todavía no hemos encontrado un valor de λ óptimo, elegimos este modelo a pesar de que tengamos un error ligeramente mayor, con la esperanza de que en la segunda fase de *Cross Validation* consigamos un error menor.

En el *dataset* al que no aplicamos PCA, es previsible que *Lasso* iba a funcionar mal. Durante el curso hemos visto la regla práctica de que, para no tener problemas de generalización, es deseable que $N > 10d_{VC}$. Tenemos, tras toda la limpieza de los datos, algo más de 15.500 columnas, y por tanto, al trabajar con modelos lineales, podemos trabajar con un número de variables en el orden de 1500 columnas. Es claro que con 81 columnas no nos interesa hacer que algunas columnas sean cero, que es lo que consigue *Lasso*.

1.3.4. *Cross-Validation, segunda etapa*

Como ya hemos justificado, elegimos regresión *Lasso* sobre el conjunto de datos al que aplicamos *PCA*. En esta fase escogemos el valor de λ: parámetro de penalización de la regularización. El rango de valores escogido es $\lambda \in [10^{-3}, 1]$, basándonos en que el el primer *step* hemos tomado $\lambda = 0{,}05$ y que estamos trabajando con una dimensionalidad alta, por lo tanto, queremos que la penalización no sea demasiado baja.

Los resultados de esta segunda fase se resumen en la siguiente fase:

Lambda	**Valor medio**	**Valor mínimo**	**Valor máximo**
0.001	-228.24	-250.70	-210.52
0.01	-228.89	-254.37	-211.02
0.1	-236.02	-261.20	-212.81
1	-319.58	-330.61	-308.83

Cuadro 7: Resultados de *Cross Validation*, segunda fase

Con los resultados de esta tabla, elegimos el parámetro del regularizador como $\lambda = 10^{-3}$, con lo que ya estamos listos para entrenar sobre todo el conjunto de datos, elegidos todos los parámetros de nuestro modelo final.

1.4. Entrenamiento sobre todo el *train_dataset* para seleccionar el modelo final

En este punto está justificado el que usemos el conjunto original de datos, al que hemos eliminado *outliers* y estandarizado. Tras esto, aplicamos *PCA* obteniendo 30 variables, y estandarizando de nuevo.. Además, usaremos como parámetros:

- $\lambda = 10^{-3}$
- Máximo de iteraciones: 1000 iteraciones
- Tolerancia: 10^{-3}

Con esto obtenemos los siguientes resultados:

Conjunto de datos	**Error cuadrático medio**	**Error absoluto medio**	R^2
Entrenamiento	216.97	10.42	0.81
Test	223.29	10.70	0.80

Cuadro 8: Resultados del entrenamiento

1.4.1. Análisis de los resultados

El error cuadrático medio viene dado por $\frac{1}{N}\sum^{N}|g(x)-y|^2$ donde g es la función aprendida, x el dato de entrada e y la etiqueta verdadera asociada. El error absoluto medio viene dado por $\frac{1}{N}\sum^{N}|g(x)-y|$, que es algo más interpretable que el error cuadrático medio. R^2 es el coeficiente de determinación lineal.

Es claro que no hemos tenido problemas de *overfitting*, pues el error en test no es demasiado dispar al error en la muestra. Los coeficientes de correlación son bastante buenos, estamos explicando el 80 % de la varianza de la muestra de test con nuestro modelo lineal.

El error absoluto medio en el test se puede interpretar como que de media, para cada dato estamos prediciendo con un error $\pm 10{,}7K$. Respecto a la bondad de nuestros resultados, en el paper original [1] se comenta que obtienen resultados con error *rmse* de $9{,}5K$, donde *rmse* es la raíz del error cuadrático medio. Nosotros obtenemos un *rmse* en test de $14{,}94K$. Por tanto, con un modelo mucho más simple que el empleado en [1], basado en árboles, hemos obtenido un *rmse* cercano a este valor de referencia de $9{,}5K$.

Además, la incertidumbre $\pm 10{,}7K$ en datos que de media tiene un $T_c = 34{,}37$ y que se mueve en un rango $T_c \in [0{,}00, 185{,}00]$ parece aceptable, aunque claramente es mejorable con el empleo de modelos más complejos, como demuestran los resultados obtenidos en [1].

Usaremos un *baseline* para ver si hemos aprendido de forma efectiva una función sobre los datos que no sea trivial. Consideramos la función *baseline* que a todo x le hace corresponder el valor medio de la variable T_c. Esto lo conseguimos con el DummyRegressor de la librería sklearn. Los resultados obtenidos son:

Conjunto de datos	**Error cuadrático medio**	**Error absoluto medio**	R^2
Entrenamiento	1173.58	29.34	0.0
Test	1172.19	29.23	-3.70e-5

Cuadro 9: Resultados del baseline usando la media

Es claro que estamos obteniendo resultados notablemente mejores, y por tanto, podemos concluir que hemos conseguido aprender de forma no trivial a partir de los datos de la muestra.

Por tanto, la principal mejora al trabajo presentado sería considerar modelos más avanzados y adecuados al tipo de problema presentado. De todas formas, queda probada la potencia de los modelos lineales, a pesar de su simplicidad, cuando se emplean el preprocesado y la transformación adecuado del conjunto de datos.

2. Problema de clasificación

2.1. Exploración del problema

Buscamos obtener información sobre distintos motores eléctricos sin necesidad de introducir más sensores. Dicha introducción de sensores provocaría un aumento de los costes, directamente por el coste de los propios sensores, e indirectamente por el coste de almacenar información que pudiera ser redundante. Por tanto, se busca clasificar motores en distintas categorías empleando información ya disponible en los motores electromecánicos, como más tarde se explicará.

Los motores se dividirán en 11 clases según los defectos que puedan presentar viendo la corriente que generan. Así que tenemos una clase para indicar estado correcto, y 10 clases para indicar distintos defectos [14].

2.1.1. Descripción del problema

Disponemos de un archivo Sensorless_drive_diagnosis .txt que contiene las 47 *features* que ya hemos indicado, y una última fila para las etiquetas de las 11 clases en las que dividimos los motores.

2.1.2. Descripción de las características

Información extraída a partir de datos otorgados por la corriente eléctrica del motor. En concreto, información extraída de las fases de la corriente eléctrica generada por el motor [14]. Estos datos pueden ser recogidos durante la conducción.

Además, los datos con los que trabajamos han sido recogidos durante distintas condiciones de operación, para cada uno de los tipos de motores.

Las *features* se extraen usando la descomposición *Empirical Model Decomposition*, que forma parte de la *Hilbert Huang Transformation*. Estos datos se vuelven a descomponer usando *Intrinsic Modal Functions*, a partir de las cuales se calculan estadísticas como media empírica, desviación estándar, oblicuidad y excedente [14].

Este proceso genera las 47 *features* que usaremos para la clasificación. Al igual que en el problema de clasificación, no disponemos de conocimiento experto para extraer, de estas 47 *features*, otras *features* de mayor calidad. Y por tanto, queda justificado que empleemos técnicas estadísticas como *PCA*.

2.1.3. Problema a resolver

Queremos aprender una función objetivo de la forma:

$$f : \mathbb{X} \to \mathbb{Y}$$

donde $\mathbb{X}$ es el conjunto de las 47 características extraídas sobre los motores, e $\mathbb{Y}$ es el conjunto $\{1, 2, \ldots, 11\}$ de todas las etiquetas en las que se agrupan los ejemplos. De nuevo, más adelante

realizaremos transformaciones sobre los datos (borrado de *outliers*, *PCA* y estandarización), con lo que realmente buscamos aprender una función objetivo de la forma $f : \hat{\mathbb{X}} \rightarrow \mathbb{Y}$ donde $\hat{\mathbb{X}} = \phi(\mathbb{X})$ es el espacio transformado del original.

Al tener una variable de salida $\mathbb{Y}$ discreta, es claro que estamos ante un problema de clasificación.

2.1.4. Descripción de las características

Ya hemos comentado el proceso de extracción de las características, a partir del proceso *EMD*. Con ello se calcula, por cada característica resultante, se calculan las estadísticas: media empírica, desviación estándar, oblicuidad y excedente. Al tener 4 estadísticas por *feature*, y al tener 47 características finales, pensamos que tenemos 11 variables originales, de las que calculamos 4 estadísticas, y otras 3 variables adicionales que suman hasta llegar a las 47 características con las que trabajamos. Esto es similar a lo que ocurría en el problema de regresión, donde la variable *number of elements* era la adicional que no iba expandida en distintas estadísticas.

2.1.5. Exploración del dataset

Antes de empezar a explorar los datos del problema, separamos el conjunto de training y de test. No queremos saber nada sobre el test dataset durante esta exploración de los datos, para evitar caer en el *data snooping*. Usamos la misma función que en regresión para llevar a cabo la separación de los datos. Sin embargo, en este caso, al tener que mantener las clases balanceadas, usamos la opción: stratify = df_Y. Podríamos pensar que con esto estamos haciendo algo de *data snooping*, pues estamos tomando información de ejemplos que acabarán en el conjunto de *test*. Sin embargo, como no vamos a hacer uso de esta información, compensa esta pequeña incorrección pues el beneficio es que nos aseguramos de no tener clases infrarrepresentadas o bien en el *test* o bien en el *training*. Lo correcto sería confiar en que, con la gran cantidad de datos que manejamos, con tomar elementos aleatorios sea suficiente.

Disponemos de un *dataset* de 58509 ejemplos. Tras la separación acabamos con 46807 para entrenamiento y 11702 para test. Como hacíamos en regresión, mostramos las estadísticas de las variables con la función explore_dataset, obteniendo la tabla *Tabla 10. Exploración estadística de los atributos del conjunto de entrenamiento*:

Columna	mean	median	var	std	min	max	p25	p75
0	-3.39e-6	-2.66e-6	6.39e-9	0.00	-0.01	0.00	-0.00	0.00
1	1.54e-6	8.95e-7	3.34e-9	0.00	-0.00	0.00	-0.00	0.00
2	1.44e-6	5.02e-7	5.65e-8	0.00	-0.01	0.00	-0.00	0.00
3	-1.38e-6	-1.06e-6	4.86e-9	0.00	-0.01	0.00	-0.00	0.00
4	1.48e-6	8.07e-7	3.32e-9	0.00	-0.00	0.00	-0.00	0.00
5	-9.27e-7	-3.05e-7	5.20e-8	0.00	-0.00	0.00	-0.00	0.00
6	1.92e-3	1.32e-2	1.32e-3	0.03	-0.13	0.06	-0.01	0.02
7	1.92e-3	1.32e-2	1.32e-3	0.03	-0.13	0.06	-0.01	0.02
8	1.91e-3	1.32e-2	1.32e-3	0.03	-0.13	0.06	-0.01	0.02
9	-1.18e-2	-1.55e-2	4.42e-3	0.06	-0.21	0.35	-0.03	0.02
10	-1.18e-2	-1.55e-2	4.42e-3	0.06	-0.21	0.35	-0.03	0.02
11	-1.18e-2	-1.55e-2	4.42e-3	0.06	-0.21	0.35	-0.03	0.02
12	1.88e-3	2.19e-3	1.13e-6	0.00	0.00	0.13	0.00	0.00
13	1.08e-3	1.18e-3	4.60e-7	0.00	0.00	0.05	0.00	0.00
14	3.09e-3	2.98e-3	4.94e-6	0.00	0.00	0.10	0.00	0.00
15	1.87e-3	2.18e-3	9.81e-7	0.00	0.00	0.10	0.00	0.00
16	1.08e-3	1.18e-3	4.46e-7	0.00	0.00	0.06	0.00	0.00
17	3.08e-3	2.90e-3	4.56e-6	0.00	0.00	0.07	0.00	0.00
18	1.61e+0	1.57e+0	1.59e-1	0.39	0.79	2.37	1.32	1.88
19	1.61e+0	1.57e+0	1.59e-1	0.39	0.79	2.37	1.32	1.88
20	1.61e+0	1.57e+0	1.59e-1	0.39	0.79	2.37	1.32	1.88
21	1.61e+0	1.57e+0	1.58e-1	0.39	0.79	2.37	1.32	1.88
22	1.61e+0	1.57e+0	1.58e-1	0.39	0.79	2.37	1.32	1.88
23	1.61e+0	1.57e+0	1.58e-1	0.39	0.79	2.37	1.32	1.88
24	1.49e-3	3.01e-3	2.76e-2	0.16 -	15.79	20.32	-0.00	0.01
25	9.68e-3	7.39e-3	6.02e-1	0.77 -	12.35	9.81	-0.20	0.22
26	-2.80e-3	2.26e-3	7.64e-1	0.87	-7.95	9.58	-0.45	0.44
27	-1.18e-4	1.41e-4	2.74e-2	0.16 -	11.90	17.22	-0.00	0.00
28	1.28e-2	8.76e-3	5.75e-1	0.75 -	12.50	10.97	-0.20	0.22
29	-8.83e-3	-4.43e-3	7.27e-1	0.85	-9.97	8.76	-0.44	0.43
30	1.70e-5	4.40e-4	6.09e-5	0.00	-0.05	0.08	-0.00	0.00
31	1.49e-5	4.38e-4	6.08e-5	0.00	-0.05	0.08	-0.00	0.00
32	2.14e-5	4.49e-4	6.08e-5	0.00	-0.05	0.08	-0.00	0.00
33	-1.73e-5	-2.68e-4	9.87e-5	0.00	-0.33	0.19	-0.00	0.00
34	-2.03e-5	-2.63e-4	9.85e-5	0.00	-0.33	0.19	-0.00	0.00
35	-1.46e-5	-2.65e-4	9.83e-5	0.00	-0.33	0.18	-0.00	0.00
36	-4.62e-1	-6.63e-1	4.38e+2 2	0.93	-0.90	4015.40	-0.71	-0.58
37	7.44e+0	3.30e+0	1.49e+2 1	2.21	-0.61	238.79	1.47	8.38
38	8.38e+0	6.54e+0	4.67e+1	6.83	0.52	125.49	4.44	9.93
39	-4.10e-1	-6.61e-1	5.72e+2 2	3.92	-0.90	3670.80	-0.71	-0.57
40	7.26e+0	3.30e+0	1.55e+2 1	2.45	-0.59	889.93	1.45	8.28
41	8.25e+0	6.47e+0	4.26e+1	6.52	0.56	153.15	4.44	9.85
42	-1.50e+0	-1.50e+0	1.34e-5	0.00	-1.52	-1.45	-1.50	-1.49
43	-1.50e+0	-1.50e+0	1.34e-5	0.00	-1.52	-1.45	-1.50	-1.49
44	-1.50e+0	-1.50e+0	1.32e-5	0.00	-1.52	-1.45	-1.50	-1.49
45	-1.49e+0	-1.49e+0	1.01e-5	0.00	-1.52	-1.33	-1.49	-1.49
46	-1.49e+0	-1.49e+0	1.01e-5	0.00	-1.52	-1.33	-1.49	-1.49
47	-1.49e+0	-1.49e+0	1.02e-5	0.00	-1.52	-1.33	-1.49	-1.49
48	6.00e+0	6.00e+0	1.00e+1	3.16	1.00	11.00	3.00	9.00

Cuadro 10: Exploración estadística de los atributos del conjunto de entrenamiento

Todas las variables son de tipo *float64*, salvo la variable de salida, que lógicamente es de tipo entero. No tenemos *missing values* así que no tenemos que preocuparnos de emplear técnicas para tratar este problema.

Todas las variables de entrada son continuas, así que tampoco es necesario emplear técnicas

como *one hot encoding* para codificar las variables categóricas.

Es destacable que tenemos distintas variables contiguas en las que las estadísticas mostradas son prácticamente la misma. Por ejemplo, las de las columnas 7 y 8, o las columnas 9, 10 y 11. Pensamos que esto es debido a que estamos usando estadísticas de una misma variable que pueden ser muy parecidas. Con ello, pensamos que aplicar la técnica *PCA* va a ser útil a la hora de reducir la dimensionalidad de nuestro problema.

De nuevo, queda clara la necesidad de una técnica como estandarización. Por ejemplo, la columna 40 tiene una desviación típica de 2.45, mientras que la columna 8 tiene una desviación típica de 0.03. O por ejemplo, la columna 36 está en un rango $[-0{,}9, 4015{,}40]$, mientras que la columna 0 está en un rango $[-0{,}01, 0]$.

Es notable que muchas columnas tienen desviación típica muy cercana a cero (algunas tienen 0.00 pero solo estamos mostrando dos posiciones decimales). Por tanto muchas de ellas estarán aportando poca información, y esperamos que *PCA* reduzca efectivamente la dimensión de nuestro espacio.

Las clases están balanceadas, como muestra la siguiente gráfica:

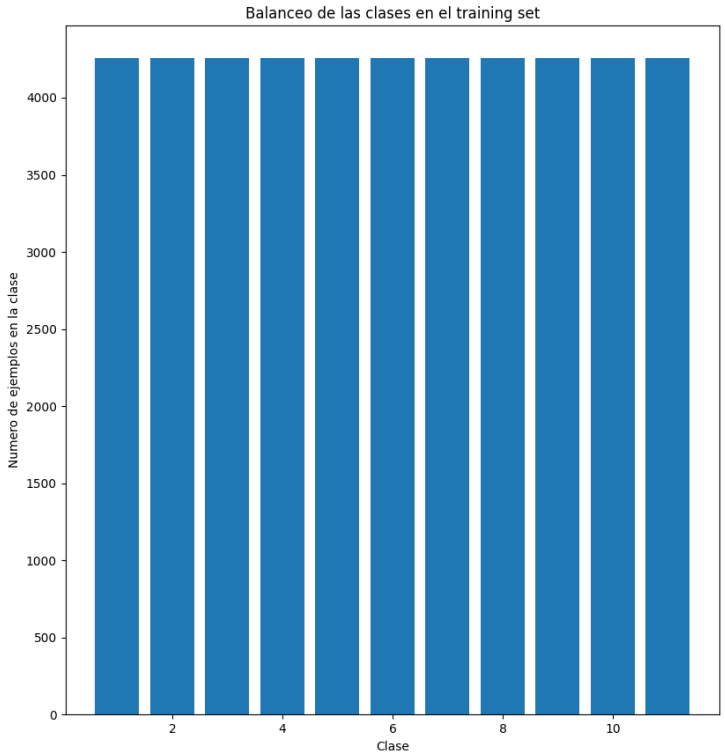

Figura 2: Gráfico de barras con el balanceo de las clases en el conjunto de entrenamiento

2.2. Preprocesado de los datos

A diferencia de lo que hacíamos en regresión, en esta sección solo vamos a mostrar los resultados del preprocesado de los datos. El eliminado de *outliers*, estandarización, uso de *PCA* y de nuevo, estandarización, ya ha sido explicado. Tanto los fundamentos teóricos, como los motivos que justifican el uso de estas técnicas, que como ya hemos visto, también son aplicables a este problema.

Tras el borrado de outliers (valores alejados de la media más de 4 veces las desviación típica), eliminamos 3039 ejemplos, quedándonos con 43768 ejemplos en el conjunto de entrenamiento. Lo que supone que hemos borrado aproximadamente 6,5 % de los datos. Un porcentaje muy parecido al borrado en regresión, así que la mejora de la calidad de los datos (estamos borrando con *z-score* 4 y no 3, que es lo usual, siendo todavía más permisivos) justifica esta pérdida de datos.

En [14], en la figura 4, se indica el accuracy obtenido en distintos modelos según se use normalización o estandarización. Para los modelos que nosotros vamos a emplear más adelante, parece claro que la mejor opción es estandarizar y no normalizar.

Así que escogemos estandarizar el *dataset* sin *outliers*. De nuevo, remarcamos que es esencial eliminar los *outliers* antes de estandarizar, pues este proceso es muy sensible a los *outliers*. Y del mismo modo, es esencial estandarizar antes de aplicar *PCA*, pues una variables con desviaciones típicas muy desbalanceadas puede provocar que unas pocas variables copen todo el protagonismo en el cambio de los vectores que forman la base del nuevo espacio.

Aplicamos la técnica *PCA*. Buscamos tener una varianza explicada por encima del 99 %. Terminamos con un espacio de dimensión 26, así que nos hemos quedado con aproximadamente el 55 % de las dimensiones iniciales. Hemos reducido la dimensionalidad del problema a casi la mitad. Esto era lo esperado por lo que ya hemos comentado: muchas variables en el mismo rango, lo que nos hacía pensar en que estaban altamente correladas, variables sin apenas varianza, ...

Como en regresión, y por los mismos motivos, aplicamos estandarización al conjunto que queda tras *PCA*. Este conjunto viene resumido en la siguiente tabla de estadísticas descriptivas:

Columna	mean	median	var	std	min	max	p25	p75
0	-6.25e-18	-0.01	1.00	1.00	-2.25	2.51	-0.63	0.61
1	1.59e-17	-0.08	1.00	1.00	-2.57	3.49	-0.55	0.35
2	-6.49e-19	0.00	1.00	1.00	-3.35	3.55	-0.71	0.72
3	-1.70e-17	0.17	1.00	1.00	-3.31	4.30	-0.80	0.68
4	-3.81e-18	-0.14	1.00	1.00	-3.99	4.44	-0.68	0.57
5	-4.95e-18	-0.05	1.00	1.00	-4.42	9.54	-0.66	0.59
6	1.86e-18	0.06	1.00	1.00	-3.21	12.39	-0.68	0.73
7	-1.29e-18	-0.00	1.00	1.00	-4.59	4.91	-0.57	0.57
8	-5.68e-18	0.00	1.00	1.00	-4.83	5.00	-0.58	0.58
9	-7.30e-18	-0.00	1.00	1.00	-5.79	6.26	-0.57	0.56
10	7.14e-18	-0.00	1.00	1.00	-5.98	6.14	-0.57	0.56
11	2.40e-17	-0.13	1.00	1.00	-3.71	12.98	-0.73	0.68
12	-5.51e-18	0.00	1.00	1.00	-6.43	8.37	-0.62	0.64
13	-3.24e-18	-0.01	1.00	1.00	-4.73	5.22	-0.52	0.51
14	-3.24e-18	-0.08	1.00	1.00	-3.32	9.62	-0.69	0.60
15	2.43e-17	0.03	1.00	1.00	-4.78	11.44	-0.63	0.66
16	-1.16e-17	0.01	1.00	1.00	-5.66	5.90	-0.35	0.36
17	-7.79e-18	-0.08	1.00	1.00	-3.62	7.30	-0.49	0.38
18	-1.78e-17	-0.02	1.00	1.00	-10.19	19.39	-0.57	0.54
19	-5.11e-18	-0.00	1.00	1.00	-5.09	7.30	-0.71	0.74
20	-1.42e-18	-0.00	1.00	1.00	-4.89	13.27	-0.58	0.57
21	1.62e-18	0.00	1.00	1.00	-4.71	5.08	-0.57	0.57
22	-2.01e-17	0.00	1.00	1.00	-15.97	29.62	-0.58	0.58
23	-5.03e-18	0.00	1.00	1.00	-6.33	6.68	-0.56	0.56
24	1.28e-17	-0.00	1.00	1.00	-20.74	33.88	-0.57	0.56
25	-2.54e-17	-0.00	1.00	1.00	-5.83	7.40	-0.48	0.46

Cuadro 11: Estadísticas descriptivas del conjunto de entrenamiento tras *PCA* y estandarización

Como era de esperar, salvo errores de precisión de punto flotante, acabamos con 26 *features* de media cero y desviación típica cero, que explican algo más del 99 % de la varianza de la muestra original. Con esto, ya estamos preparados para realizar la selección del modelo empleando la técnica de *Cross Validation*

2.3. Selección del modelo

En este caso, solo vamos a usar una vez validación cruzada, a diferencia de regresión en el que hacíamos dos fases. Los modelos considerados serán Regresión Logística y *Soft Support Vector Machine*.

El modelo de regresión logística es muy parecido al del perceptrón, solo que cambiamos la función de activación. En el perceptrón, tomamos la señal dada por $\sum w_i x_i$ y le aplicamos la función de activación identidad. En regresión logística aplicamos la función de activación σ.

Consideramos este modelo porque, como se ha desarrollado en la teoría de este curso, este modelo funciona muy bien cuando hay cierto *overlapping* entre las clases con las que trabajamos. Podemos pensar que tenemos cierto *overlapping* debido a que distintos defectos puedan tener *"síntomas"* en la señal eléctrica muy parecidos.

Consideramos también *Soft Support Vector Machine*. Es un modelo muy potente gracias al uso del *kernel trick*, que permite transformaciones de los datos a espacios de dimensión elevada sin mucho coste. Además, con la maximización del margen y el hecho de que la solución final solo depende de unos pocos vectores soporte, podemos usar estas altas dimensionalidades sin tanto riesgo de caer en el *overfitting*.

En regresión logística consideramos el conjunto sin transformar y el conjunto transformado, al igual que en regresión, con monomios de hasta orden 2. No podemos computar transformaciones con monomios de mayor grado, y por tanto no las consideramos. En *SVM*, las transformaciones las especificaremos con el kernel dado por parámetro.

En ambos casos moveremos el valor de C que especificará la inversa del parámetro de regularización.

En regresión logística consideramos los siguientes parámetros:

- Usamos regularización $L2$, la que usábamos en el modelo lineal *Ridge*. No tenemos ningún motivo fuerte para emplear *Lasso* pues no nos interesa realizar selección de características. Tenemos pocas características y solo estamos considerando transformaciones polinómicas de hasta grado 2
- Regresión logística es un modelo para clasificación binaria. Por tanto debemos usar alguna técnica para adaptarlo a clasificación multiclase. La técnica escogida es multinomial. Con ello, se busca minimizar la función de pérdida multinomial sobre toda la distribución de las variables [15]
- Usamos el valor por defecto para el máximo de iteraciones: 100. Usando este valor tenemos buenos resultados, como se mostrará más adelante, en unos tiempos razonables

En *SVM* consideraremos los siguientes parámetros:

- kernel: probaremos con kernel lineal, polinómico (en cuyo caso variaremos el grado de la transformación) y *radial basis function*.
- La regularización será obligadamente por sklearn *l2*.
- No consideraremos número máximo de iteraciones. Consideraremos una tolerancia umbral entre dos iteraciones de 10^{-3}, valor correspondiente al parámetro por defecto

- De nuevo, *SVM* es un clasificador binario. Con el parámetro decision_function_shape=" ovr" especificamos que queremos usar la técnica *one versus rest*.

En ambos casos, estamos aplicando *k-fold cross validation* con un valor $k = 10$. Y en ambos casos, como métrica estamos usando el *accuracy* pues es muy fácil de interpretar a la hora de quedarnos con el modelo que mayor valor de esta métrica presente.

2.3.1. Cross Validation

De nuevo, el código que implementa todo lo que ya hemos comentado se encuentra en la función show_cross_validation.

Esta función consume muchísimo tiempo. Por ello, no hemos sido capaces de terminar de probar con todos los modelos que se habían propuesto inicialmente. Paro la ejecución tras seis horas de cómputo en mi máquina [2]. Por tanto, el código que llama a show_cross_validation está comentado.

Los resultados obtenidos se muestran en la *Tabla 12. Resultados del proceso de* Cross Validation

Es claro que los dos modelos que mejor media de *acuraccy* son Regresión Logística, con una transformación cuadrática, y valor de C 1.05 y 1.20. Elegimos $C = 1{,}20$ pues el rango $[min, max]$ tiene una longitud de 0.05, mientras que en el otro caso tenemos una longitud de 0.09, y así el valor $C = 1{,}20$ pare ser más estable.

Los resultados para los parámetros que nos ha dado tiempo a ejecutar en *Support Vector Machine* son muy buenos, pero no han logrado el nivel de regresión logística.

[2] En *4. Consideraciones adicionales* mostramos las características de la máquina en la que hemos realizado la ejecución

Modelo	Transformación	C	Media	Mínimo	Máximo
LogReg	Polinomio orden 1	0.9	0.918	0.915	0.923
LogReg	Polinomio orden 1	0.95	0.918	0.912	0.925
LogReg	Polinomio orden 1	1.0	0.918	0.911	0.922
LogReg	Polinomio orden 1	1.05	0.919	0.912	0.923
LogReg	Polinomio orden 1	1.1	0.918	0.913	0.923
LogReg	Polinomio orden 1	1.15	0.919	0.913	0.925
LogReg	Polinomio orden 1	1.2	0.918	0.914	0.926
LogReg	Polinomio orden 2	0.9	0.964	0.961	0.970
LogReg	Polinomio orden 2	0.95	0.964	0.960	0.968
LogReg	Polinomio orden 2	1.0	0.964	0.958	0.967
LogReg	Polinomio orden 2	1.05	0.965	0.959	0.968
LogReg	Polinomio orden 2	1.1	0.964	0.959	0.968
LogReg	Polinomio orden 2	1.15	0.964	0.961	0.971
LogReg	Polinomio orden 2	1.2	0.965	0.962	0.967
SVC	Kernel Lineal orden 1	0.9	0.929	0.927	0.933
SVC	Kernel Lineal orden 1	0.95	0.929	0.924	0.935
SVC	Kernel Lineal orden 1	1.0	0.929	0.925	0.933
SVC	Kernel Lineal orden 1	1.05	0.930	0.920	0.935
SVC	Kernel Lineal orden 1	1.1	0.929	0.924	0.935
SVC	Kernel Lineal orden 1	1.15	0.929	0.925	0.933
SVC	Kernel Lineal orden 1	1.20	0.929	0.923	0.937
SVC	Kernel Lineal orden 2	0.9	0.929	0.923	0.934
SVC	Kernel Lineal orden 2	0.95	0.929	0.927	0.932
SVC	Kernel Lineal orden 2	1.0	0.929	0.922	0.933
SVC	Kernel Lineal orden 2	1.05	0.929	0.924	0.934
SVC	Kernel Lineal orden 2	1.1	0.929	0.925	0.937
SVC	Kernel Lineal orden 2	1.15	0.929	0.925	0.932
SVC	Kernel Lineal orden 2	1.2	0.929	0.922	0.934
SVC	Kernel Lineal orden 3	0.9	0.929	0.926	0.932
SVC	Kernel Lineal orden 3	0.95	0.929	0.924	0.934
SVC	Kernel Lineal orden 3	1.0	0.929	0.925	0.936
SVC	Kernel Lineal orden 3	1.05	0.929	0.918	0.934
SVC	Kernel Lineal orden 3	1.1	0.929	0.925	0.932
SVC	Kernel Lineal orden 3	1.15	0.929	0.923	0.935
SVC	Kernel Lineal orden 3	1.20	0.929	0.924	0.936

Cuadro 12: Resultados del proceso de *Cross Validation*

3. Entrenamiento sobre todo el *training_dataset* para seleccionar el modelo final

En este punto está justificado que usemos el conjunto original de datos, al que hemos eliminado *outliers* y estandarizado. Tras esto, aplicamos *PCA* obteniendo 26 variables, y estandarizando de nuevo. Además, usamos como parámetros:

- Regularización *l2*
- Clasificación multiclase usando LGR multinomial
- Máximo de iteraciones: 100
- Valor de la inversa del regularizador: $C = 1{,}20$

Mostramos los resultados en la siguiente tabla:

Muestra	**Accuracy**	**Precisión**	**F1**
Entrenamiento	0.983	0.983	0.983
Test	0.947	0.947	0.947

Cuadro 13: Resultados del entrenamiento

3990	1	0	0	0	10	0	0	0	0	0
0	3740	0	0	0	0	0	0	0	212	1
0	1	3988	0	8	0	0	1	0	0	0
0	0	1	3982	2	0	0	0	0	0	0
0	0	9	1	3892	2	0	129	1	0	0
18	1	0	0	0	3921	0	2	12	0	0
0	0	0	0	0	0	4018	0	0	0	0
0	0	0	0	127	3	0	3847	0	0	0
0	1	0	0	1	21	0	0	3943	0	0
0	164	1	0	0	0	0	0	1	3763	0
0	0	0	0	0	0	0	0	0	0	3953

Cuadro 14: Matriz de confusión en la muestra de entrenamiento

1041	2	3	1	1	25	1	2	0	5	0
0	965	1	0	5	3	1	1	5	88	29
1	10	1028	10	16	9	3	3	3	2	0
0	0	1	1028	10	0	5	4	1	0	0
0	1	19	21	966	2	2	43	6	2	0
21	1	0	0	0	994	0	2	11	0	0
0	0	0	1	0	0	1052	0	0	0	2
0	0	0	1	64	7	0	996	2	0	1
1	10	8	1	2	24	0	13	1035	2	11
0	74	4	1	0	0	0	0	1	965	0
0	0	0	0	0	0	0	0	0	0	1020

Cuadro 15: Matriz de confusión en la muestra de test

3.0.1. Análisis de los resultados

El estadístico *Accuracy* se calcula como el porcentaje de valores bien predichos entre todos los valores sobre los que realizamos el cálculo. La *precisión* es el porcentaje de verdaderos positivos entre verdaderos y falsos positivos. Como estamos trabajando con clasificación multiclase, calculamos este estadístico clase a clase y calculamos la media entre todas las clases. Esto usando la opción de sklearn average = 'macro'. *F1* es la media armónica entre precisión y *recall*, donde *recall* es el porcentaje de verdaderos positivos que clasificamos correctamente. De nuevo, debemos usar la técnica average = 'macro' [16].

Lo primero es observar que estamos obteniendo unos resultados muy buenos en el test, con las tres estadísticas entorno al 94,7 % (cuando mostramos más de 3 decimales, vemos que no son exactamente el mismo valor). Son estadísticas que no se alejan demasiado (en torno a un 4 %) de las estadísticas logradas en el conjunto de entrenamiento. Por tanto, no hemos tenido problemas de *overfitting*.

En segundo lugar, la matriz de confusión en la muestra de test muestra también buenos resultados. En esta matriz, la fila i indica el valor verdadero de la clase, y la fila j muestra el valor en el que hemos clasificado. Por tanto, lo que queremos es que la mayor parte de valores estén en la diagonal principal, como es nuestro caso. Vemos que en algunas clases estamos clasificando mal, por ejemplo, confundimos la clase 2 con la clase 10 en 88 ejemplos de la muestra. Sin embargo, teniendo en cuenta que estamos trabajando con una muestra de test con 11702 ejemplos, estos valores que quedan fuera de la diagonal principal no parecen relevantes. Ninguno de estos valores fuera de la diagonal llega a los 100 ejemplos, y por tanto, en una muestra de 11700 ejemplos no parecen relevantes.

Es decir, hemos aprendido efectivamente una función que aproxima f de forma no trivial (por ejemplo, una función que siempre clasifique en la misma etiqueta, o que clasifique aleatoriamente). Además, parece más complejo aprender una función trivial en clasificación multietiqueta que alcance un *performance tan bueno*.

El principal problema con el que nos hemos encontrado ha sido el tiempo de cómputo necesario en *Cross Validation* con *SVM*. Por tanto, la principal mejora que podemos considerar es intentar solucionar esto. Una forma que pensamos que puede ser efectiva es aplicar *PCA* para acabar con menos de 26 dimensiones, a pesar de perder algo de varianza explicada, con la esperanza de acelerar el proceso de aprendizaje para *SVM* manteniendo buenos resultados. O bien usar algún servicio en la nube como *Google Collab* para intentar realizar un entrenamiento con servidores más potentes. A pesar de esto, hemos obtenido unos resultados muy buenos empleando un modelo lineal como es *Regresión Logística*.

En [14], figura 7, se muestra el *performance* de los clasificadores entrenados. Con *SVM* logran un *accuracy* del 99 %, por lo tanto, las mejoras que proponemos anteriormente tienen una motivación clara en base a estos resultados. Sospechamos que, con un equipo más potente, podemos conseguir resultados mucho más buenos aplicando el *kernel* adecuado.

Sin embargo, en [14] muestran como usando una transformación *RFE*, se logran resultados mucho mejores que empleando *PCA*. Por tanto, tampoco tenemos una confianza total en que usando *SVM* sobre una máquina más potente obtengamos resultados similares a los presentados en [14]. Durante esta práctica hemos comprobado empíricamente le gran impacto que tiene un buen preprocesado de los datos. Por tanto, los buenos resultados obtenidos en [14] seguramente se fundamenten en gran medida en un buen preprocesado de los datos, que con *PCA* justifican que no se logra.

4. Consideraciones adicionales

Estamos usando la librería scipy para el borrado de los outliers. En concreto, para hacer una *query* a pandas en la que calculamos el *z-score*. Esto se ve en la función remove_outliers

Por otro lado, las características de la máquina en la que ejecutamos el código se muestra con la traza del comando lscpu

```
Architecture:                    x86_64
CPU op-mode(s):                  32-bit, 64-bit
Byte Order:                      Little Endian
Address sizes:                   39 bits physical, 48 bits virtual
CPU(s):                          4
On-line CPU(s) list:             0-3
Thread(s) per core:              2
Core(s) per socket:              2
Socket(s):                       1
NUMA node(s):                    1
Vendor ID:                       GenuineIntel
CPU family:                      6
Model:                           142
Model name:                      Intel(R) Core(TM) i7-7500U CPU @
    2.70GHz
Stepping:                        9
CPU MHz:                         2432.428
CPU max MHz:                     3500,0000
CPU min MHz:                     400,0000
BogoMIPS:                        5802.42
Virtualization:                  VT-x
L1d cache:                       64 KiB
L1i cache:                       64 KiB
L2 cache:                        512 KiB
L3 cache:                        4 MiB
```

Además, la máquina tiene 8GB de memoria RAM.

5. Referencias

[1] K. Hamidieh, "A data-driven statistical model for predicting the critical temperature of a superconductor," *Elsevier Computational Materials Science*, 2018.

[2] "6.3. preprocessing data — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/preprocessing.html#scaling-data-with-outliers. (Accessed on 28/05/2021).

[3] "Pca: Application in machine learning — by harsha goonewardana — apprentice journal — medium." https://medium.com/apprentice-journal/pca-application-in-machine-learning-4827c07a61db. (Accessed on 31/05/2021).

[4] "Feature scaling — standardization vs normalization." https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/. (Accessed on 03/06/2021).

[5] "sklearn.preprocessing.standardscaler — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html. (Accessed on 03/06/2021).

[6] "Principal component analysis - wikipedia." https://en.wikipedia.org/wiki/Principal_component_analysis. (Accessed on 02/06/2021).

[7] "Principal component analysis: a review and recent developments — philosophical transactions of the royal society a: Mathematical, physical and engineering sciences." https://royalsocietypublishing.org/doi/10.1098/rsta.2015.0202. (Accessed on 02/06/2021).

[8] "normalization - why do we need to normalize data before principal component analysis (pca)? - cross validated." https://stats.stackexchange.com/questions/69157/why-do-we-need-to-normalize-data-before-principal-component-analysis-pca. (Accessed on 03/06/2021).

[9] "3.3. metrics and scoring: quantifying the quality of predictions — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter. (Accessed on 03/06/2021).

[10] "sklearn.preprocessing.polynomialfeatures — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html. (Accessed on 03/06/2021).

[11] "sklearn.linear_model.linearregression — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html#sklearn.linear_model.LinearRegression. (Accessed on 03/06/2021).

[12] "sklearn.linear_model.ridge — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge. (Accessed on 03/06/2021).

[13] "1.1. linear models — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/linear_model.html. (Accessed on 03/06/2021).

[14] T. G. et al., "Evaluation of machine learning for sensorless detection and classification of faults in electromechanical drive system," *Elsevier*, 2020.

[15] "sklearn.linear_model.logisticregression — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html#sklearn.linear_model.LogisticRegression. (Accessed on 04/06/2021).

[16] "The 5 classification evaluation metrics every data scientist must know — by rahul agarwal — towards data science." https://towardsdatascience.com/the-5-classification-evaluation-metrics-you-must-know-aa97784ff226. (Accessed on 04/06/2021).