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

3 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 caracteríticas.*

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 caracteríticas*, 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 $\hat{\mathcal{X}}$ tendra 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 caracteríticas

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	Energia necesaria para eliminar una valencia del electrón
Radio Atómico	Radio atómico
Densidad	Densidad a una temperatura y presión estandar
Afinidad del electrón	Energia 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 estandar
- Desviación estandar 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 caracteríticas. 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*:

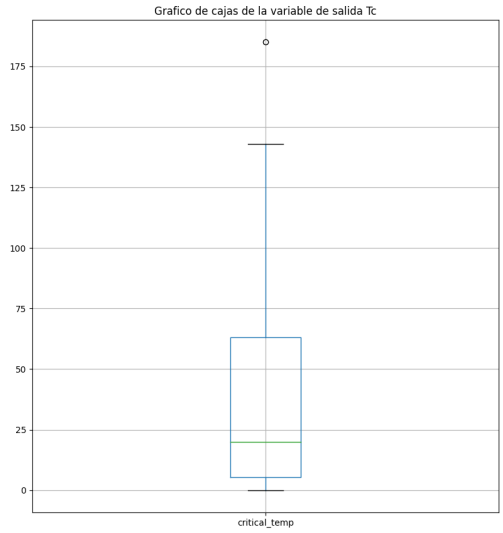

Figura 1: Boxplot de la temperatura crítica

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

1.2.2. 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 [4] [5]. 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 trasnformación 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.

Cuando buscamos un 95 %, con 3 componentes parece que es suficiente. Esto parece tener 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. Sin embargo, quedarnos de un conjunto de 81 variables con solo 3 parece excesivo. Por tanto tomamos arbitrariamente la decisión de quedarnos con 10 columnas, el 12.3 % de las variables. Esto porque ya hemos comentado que tenemos 8 variables atómicas, y añadimos dos variables más para tener algo de margen con las estadísticas moleculares.

Quedándonos con 10 variables obtenemos un 99.950 % de la variabilidad original explicada.

Un detalle destacable es que antes de aplicar *PCA* no estandarizamos el conjunto de datos, pues esto produce peores resultados. Esto tiene sentido pues *PCA* se basa en correlación de entre dos variables, que es alterada cuando hacemos estandarización basada en una única variable a la vez (como veremos más adelante).

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	-2.28e-13	-3464.06	4.79e+7	6923.94	-8121.31	39063.60	-4783.59	3770.72
1	2.05e-12	10.68	2.30e+7	4798.68	-11192.14	18391.68	-2274.11	1340.20
2	-3.15e-13	-17.09	3.82e+6	1956.17	-10235.37	14248.53	-774.02	684.64
3	-6.75e-13	98.48	8.06e+5	897.95	-3794.20	7688.15	-574.11	512.52
4	-8.79e-13	93.72	6.67e+5	816.88	-4432.94	5548.37	-539.45	479.13
5	6.42e-13	-82.54	4.10e+5	641.01	-2259.87	4500.24	-273.61	286.83
6	-4.11e-13	-3.76	1.16e+5	340.72	-1790.88	2917.67	-182.77	175.26
7	-9.67e-14	-49.58	6.11e+4	247.32	-1530.81	2782.81	-153.15	135.49
8	6.75e-15	7.06	3.34e+4	182.90	-779.76	855.90	-79.61	111.87
9	1.09e-13	-8.45	1.61e+4	127.25	-510.65	1033.12	-73.78	66.47

Cuadro 3: 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.

1.2.3. 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 [6]. En normalización conseguríamos 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 [7].

Aplicamos estandarización tanto al conjunto original de datos, al que solo hemos borrado outliers, como al conjunto que queda tras aplicar *PCA*.

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, de ambos conjuntos de datos tras estandarizar.

name	**mean**	**median**	**var**	**sdt**	**min**	**max**	**p25**	**p75**
0	-1.37-17	-0.50	1.00	1.00	-1.17	5.64	-0.69	0.54
1	-1.67-17	0.00	1.00	1.00	-2.33	3.83	-0.47	0.27
2	-3.44-18	-0.00	1.00	1.00	-5.23	7.28	-0.39	0.35

Cuadro 4: Conjunto de datos tras aplicar *PCA* y *estandarización*

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 5: Conjunto de datos sin aplicar *PCA* tras la *estandarización*

En ambas 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 5. 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.

1.3. 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 mética 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 [8].

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 trasnformaciones polinómicas
- El algoritmo de aprendizaje: mínimos cuadrados ordinarios, Ridge o Lasso

Respecto a las trasnformaciones de los datos, a falta de conocimiento experto sobre el problema para guiar las trasnformaciones empleadas, probamos con trasnformaciones 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 [9].

El primer modelo consiste en minimizar el error cuadrático medio a través de la matriz pseudoinversa [10]. 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 [11]:

- 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 [12].

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 tuviésemnos que elegir solo uno, eligiríamos *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, dandole 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 polinímicas en el conjunto $\{1, 2, 3, 4\}$. Al conjunto de datos al que no aplicamos *PCA*, no aplicamos transformaciones polinómicas, pues nuestro ordenador no es capaz de calcular estas transformaciones.

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	-572.19	-601.20	-535.89
Lineal	PCA	2	-431.65	-476.37	-412.32
Lineal	PCA	3	-350.63	-386.93	-326.35
Lineal	PCA	4	-2470.37	-15358.09	-283.22
Ridge	PCA	1	-572.15	-609.16	-541.24
Ridge	PCA	2	-431.82	-472.09	-404.95
Ridge	PCA	3	-348.22	-363.54	-332.83
Ridge	PCA	4	-2126.95	-14843.43	-333.92
Lasso	PCA	1	-572.15	-634.92	-549.44
Lasso	PCA	2	-432.08	-459.59	-413.28
Lasso	PCA	3	-358.46	-382.28	-335.32
Lasso	PCA	4	-361.25	-592.61	-304.22
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 es mejor no aplicar *PCA* y transformaciones polinómicas. Quizás con otra transformación sí que sería útil aplicar *PCA*, pero en este caso no hemos encontrado unas trasnformaciones útiles. Centrándonos en las tres variantes para el *dataset* sin aplicar *PCA*, el mejor modelo es *Linear Regression*. Sin embargo, *Ridge* pierde en media por un error cuadrático medio de tan solo 0,27. 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*. Lo que sí nos interesa es que los valores de los parámetros del modelo no sean muy grandes, que es lo que consigue *Ridge*.

1.3.4. *Cross-Validation, segunda etapa*

Como ya hemos justificado, elegimos regresión *Ridge* sobre el conjunto de datos al que no aplicamos *PCA*. Recordar que, a pesar de no estar aplicando *PCA*, si estamos aplicando el borrado de *outliers* y la estandarización de los datos.

En esta fase escogemos el valor de λ: parámetro de penalización de la regularización. Estamos trabajando con solamente 81 columnas, con un conjunto de algo más de 15.500 ejemplos de entrenamiento. Por tanto, podemos esperar que acabemos con un valor de λ pequeño. Por tanto, el rango de valores escodigo es $\lambda \in [10^{-7}, 5]$.

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

Lambda	**Valor medio**	**Valor mínimo**	**Valor máximo**
1e-07	-310.50	-322.55	-282.40
1e-06	-310.50	-322.55	-282.40
5e-06	-310.55	-335.61	-272.69
1e-05	-310.77	-335.67	-292.98
0.0001	-310.74	-343.24	-291.18
0.001	-310.79	-348.74	-287.12
0.01	-310.58	-356.06	-283.69
0.05	-310.58	-338.59	-280.38
0.1	-310.75	-324.71	-298.66
1	-310.99	-335.25	-276.35
2	-311.13	-329.18	-291.28
5	-311.90	-336.92	-287.86

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

Los resultados de esta tabla eran previsibles por lo que ya hemos comentado. El error aumenta monótamente salvo en el salto de $\lambda = 0{,}001$ a $\lambda = 0{,}01$. El mejor resultado lo obtenemos con $\lambda \in \{10^{-7}, 10^{-6}\}$. Elegimos por tanto 10^{-6}, con 10^{-7} puede ser que estemos castigando muy poco en la parte de regularización.

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, sin emplear una técnica como *PCA*. Además, usaremos como parámetros:

- $\lambda = 10^{-6}$
- 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	307.77	13.26	0.73
Test	311.28	13.40	0.73

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 buenos, estamos explicando el 73 % de la varianza de la muestra 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 $\pm 13{,}4K$. 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 $17{,}64K$. Por tanto, con un modelo mucho más simple que el empleado en [1], basado en árboles, hemos obtenido un *rmse* de menos del doble.

Además, la incertidumbre $\pm 13{,}4K$ 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 muy mejorable.

El principal problema con el que nos hemos encontrado ha sido realizar una reducción de la dimensionalidad de nuestro conjunto de datos, acompañada con una transformación no lineal de los datos, que provocase un aumento del performance de la función aprendida sin caer en problemas de generalización. Como se ha justificado durante las secciones anteriores, *Cross Validation* ha mostrado que nuestra transformación de los datos no ha sido efectiva, usando prácticamente los datos tal y como se nos han dado (previo borrado de *outliers* y estandarización obligada).

==¿TODO – en algun commento comentar que estamos usando unos baselines

2. Problema de clasificación

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

[5] "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).

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

[7] "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).

[8] "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).

[9] "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).

[10] "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).

[11] "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).

[12] "1.1. linear models — scikit-learn 0.24.2 documentation." https://scikit-learn.org/stable/modules/linear_model.html. (Accessed on 03/06/2021).