Práctica 3

Ajuste de datos usando modelos lineales

30 de mayo de 2021

Aprendizaje Automático

FRANCISCO JAVIER SÁEZ MALDONADO

fjaviersaezm@correo.ugr.es

Índice

Introducción

En esta práctica, trataremos de realizar un estudio completo de un problema en el que se nos presenta un conjunto de datos y nuestro objetivo es seleccionar el mejor predictor lineal para este conjunto de datos dado. Concretamente, estudiaremos dos conjuntos de datos extraidos de la web UCI-Machine Learning Repository.

Utilizaremos uno de ellos para tratar de ajustar un modelo lineal a un problema de regresión, y otro conjunto diferente de datos para ajustar otro modelo lineal a un problema de clasificación multiclase. El objetivo será realizar un estudio de los datos, evitando en todo momento el *data snooping*, y argumentar si se utilizan ciertas técnicas de preprocesado de datos antes de escoger el modelo final.

Trataremos primero el problema de regresión y posteriormente el de clasificación.

1. Regresión

1.1. Estudio del conjunto de datos. Identificación de $\mathcal{X}, \mathcal{Y}, f$.

Lo primero que debemos hacer es realizar una buena comprensión de la información que tenemos sobre los datos para comprender un poco más nuestro problema.

Nuestro primer conjunto de datos, [2], contiene características de ciertos elementos superconductores. Junto con estas características, se nos presenta una *temperatura crítica*, que en [1] lo denominan T_c siendo un problema similar al que vamos a abordar, obtenida para un superconductor que posea estas características. También se nos presenta un archivo en el que se nos dan las fórmulas químicas de los superconductores, pero este archivo no será relevante para nosotros.

Las características que obtenemos para este problema han sido generadas utilizando diferentes técnicas aplicadas a cada dato que se tenía inicialmente. Algunos de estos datos son su masa atómica, la energía requerida para ionizar el átomo, la densidad, la afinidad a nuevos electrones, temperatura de fusión, conductividad termal y la valencia del compuesto. Usando estos datos, se realizan una serie de transformaciones sobre estos valores para obtener el conjunto de datos final, limpiándolos durante el proceso de preparación, lo cual nos da unos datos con pocos errores o datos inútiles (se eliminan repetidos o aquellos que tengan $T_c = 0$).

Lo primero que nos encontramos acerca de nuestros datos es la siguiente tabla:

Características	Multivariable	Número de instancias	21263
Tipo de características	Reales	Número de atributos	81
Tareas asociadas	Regresión	Valores perdidos	$N \backslash A$

Tabla 1: Datos contenidos en el conjunto de datos Superconductivity.

Esta información nos resulta muy útil, pues obtenemos podemos observar que tenemos 81 atributos para cada una de las 21263 instancias. De aquí podemos obtener que el tamaño del conjunto de datos es bastante amplio, por lo que tendremos un buen conjunto de entrenamiento. Las características que obtenemos son reales, es decir, $x_i \in \mathbb{R}^{81}$. Para completar, vemos que no tenemos valores perdidos, por lo que nos ahorraremos en este caso tener que establecer una técnica para reconstruir estos valores.

Con la información proporcionada podemos decir que:

1. Nuestro conjunto de datos de entrenamiento será

$$\mathcal{X} = \{x_i \in \mathbb{R}^{81}, \text{ con } i = 1, \ldots, 21263\}.$$

2. Nuestro conjunto de etiquetas, puesto que no se nos indica ninguna restricción sobre las temperaturas, podemos asumir que es:

$$\mathcal{Y} = \{y_i \in \mathbb{R}, \text{ con } i = 1, \ldots, 21263\}.$$

3. Por último, nuestra función $f : \mathcal{X} \to \mathcal{Y}$ que asigne a cada vector de características una temperatura crítica.

Hay que anunciar que los siguientes gráficos de visualizado de datos se han realizado posteriormente a realizar la separación en conjuntos de *train* y *test* de nuestro conjunto de datos, para evitar en todo momento el *data snooping*.

Dibujamos ahora un gráfico en el que mostramos el diagrama de caja de los posibles valores que toma la temperatura T_c:

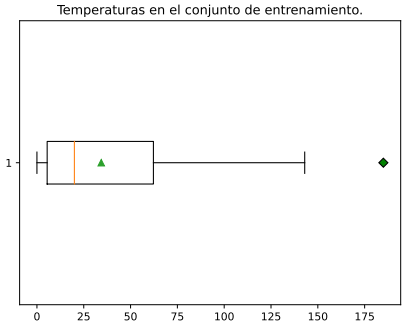

Figura 1: Diagrama de caja de las temperaturas T_c en el conjunto de entrenamiento.

Podemos ver que tenemos una variabilidad razonablemente amplia en los valores que toma f. Sin embargo, se observa que la mayoría de los valores de f están concentrados en el intervalo $[0, 50]$, pero también tenemos valores que se alejan bastante de este intervalo. Esto nos podría indicar que nuestra muestra está sesgada, en el sentido de que no tenemos muchos puntos x_i en nuestro dataset que nos den valores altos de la temperatura T_c. Como podemos ver, tenemos un dato que se aleja mucho de 1.5 por el rango intercuartílico, que es lo que representan los *bigotes* del diagrama de caja. Es por ello que podemos decir que este punto es posiblemente un *outlier*.

Además, se ha tratado de encontrar si hay características que ofrezcan una desviación típica muy baja y que por ello pudieran no tener utilidad a la hora de entrenar nuestro modelo o hacer cálculos. Sin embargo, hemos encontrado que no hay ninguna característica con una desviación típica menor de 0.05, por lo que no eliminamos por este criterio ninguna columna de nuestros datos.

1.2. La clase de funciones $\mathcal{H}$. Las hipótesis finales.

La clase de funciones a utilizar en este caso viene impuesta por el enunciado del ejercicio. En este caso, utilizaremos la clase de las funciones lineales:

$$\mathcal{H} = \{h(x) = w^T x \ : \ w \in R^{n+1}\}.$$

Tenemos que destacar que, aunque se podría plantear aplicar funciones no lineales a las características dadas (ejemplo $\phi : \mathbb{R}^d \to \mathbb{R}^k$ dada por $\phi(x) = (1, x_1, \cdots, x_n, x_1x_1, x_1x_2, \cdots, x_dx_d)$, no se hace en este caso pues estaríamos añadiendo ua complejidad a la clase de funciones sin saber realmente si esto sería útil de cara a la generalización o no. Es por ello que, al no tener en la información sobre los datos que se

nos proporciona ningún motivo para hacerlo, se decide no aplicar ninguna transformación de este estilo a los datos.

Una vez fijada la clase de funciones, el modelo que usaremos para este problema es regresión lineal. No tiene sentido utilizar otros métodos como perceptron pues se usan en problemas de clasificación.

Además, hay que comentar que para realizar esta regresión se utilizará el algoritmo de gradiente descendente estocástico (SGD), pues es bastante eficiente, unido a que podemos encontrar la implementación de esta regresión usando SGD en sklearn.

1.3. Conjuntos de entrenamiento, validación y test.

En este problema, tenemos un conjunto suficientemente grande de datos, que no viene previamente separado en subconjuntos de entrenamiento y test. En concreto, hemos mencionado ya que $N = 21263$ datos. Es por ello que se ha decidido usar un conjunto de entrenamiento con el 70 % de los datos, y dejar el 30 % para el conjunto de test. Para ello nos aprovechamos de la función `train\_test\_split` de `sklearn`.

Para elegir en nuestro conjunto de hipótesis antes de evaluar la función elegida en el conjunto de test, utilizaremos la conocida técnica **K-Fold Cross Validation**.

Esta técnica consiste en, si llamamos X_{train} al conjunto de entrenamiento, realizar los siguientes pasos:

Algorithm 1 K-Fold Cross Validation

1: $Vector\_Eouts = []$
2: **for** i = 1,...,k **do**
3: $\quad Datos_{val} \leftarrow Particion_i$
4: $\quad Datos_{train} \leftarrow X_{train} \backslash Particion_i$
5: $\quad Pesos \leftarrow$ Entrenamiento en $Datos_{train}$
6: $\quad Vector_Eouts \leftarrow Error(Pesos, Datos_{val})$
7: **end for**
8: **return** Average $Vector\_Eouts$

Describiéndolo en pocas palabras, diríamos que partimos el conjunto de entrenamiento en k subconjuntos y en cada iteración entrenamos nuestro modelo con $k-1$ particiones y calculamos el error "fuera de la muestra" (lo llamamos así porque lo calculamos sobre el conjunto de datos de entrenamiento que **no** hemos usado para entrenar) usando la partición restante. Hacemos eso con todas las particiones y devolvemos una media de los errores fuera de la muestra que hemos obtenido. Les decimos errores fuera de la muestra porque se calculan usando puntos no usados **en esa iteración del entrenamiento**, aunque formen parte del conjunto de datos.

Obteniendo el error medio en validación usando este tipo de validación cruzada, podemos hacernos una idea de cómo de bueno (en media) será nuestro modelo fuera de la muestra. De hecho, sabemos que:

Teorema.- El error de validación cruzada E_{cv} es un estimador insesgado de la esperanza del error fuera de la muestra en conjuntos de datos de tamaño $N-1$.

Usualmente, $K-$Fold cross validation se utiliza para estimar los parámetros con los que se entrenará nuestro modelo final, y una vez que se han estimado, se vuelve a entrenar el modelo usando todos los datos de entrenamiento disponible para tener un modelo entrenado con un conjunto de datos lo mayor posible.

En nuestro caso, usaremos `StratifiedKFold` de *sklearn*. Esta función nos devuelve el número de folds k que le indiquemos, con la salvedad de que se intenta mantener la distribución de datos existente en el conjunto en cada uno de los subconjuntos. En el caso de regresión, en cada partición obtenida tendremos valores de f distribuidos como los tenemos en el conjunto de entrenamiento completo.

1.4. Preprocesado de datos.

Entramos en una de las fases más importantes de nuestro problema. Vamos a ver qué transformaciones haremos sobre nuestros datos antes de realizar la regresión.

En cuanto a los valores de nuestros datos, si tomamos una media de las desviaciones típicas σ de los atributos de cada elemento de nuestro conjunto de datos, el resultado es:

```
Average Standard deviation of the features of the dataset per row: 1613.81306
```

Por lo que obtenemos que claramente los valores de los diferentes atributos no están en el mismo rango de escala. Es por ello que previamente al entrenamiento realizaremos una estandarización por atributos de nuestro conjunto de datos. Esto nos permitirá que sean comparables entre ellos.

Recordamos que tenemos 81 variables para cada dato. Nos interesa saber si todas estas variables son completamente útiles para el entrenamiento o nos interesa hacer una reducción de dimensionalidad en nuestro problema. Vamos a hacer una visualización las correlaciones entre las características para ver si algunas de ellas están altamente correladas.

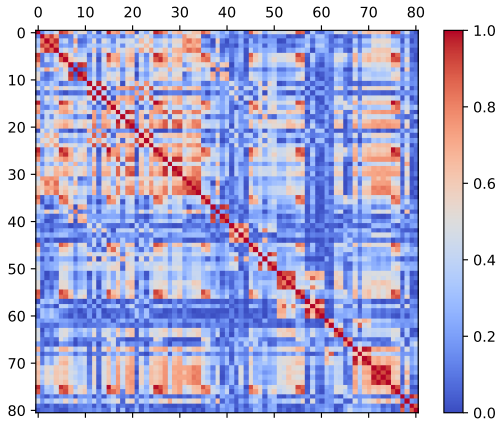

Figura 2: Matriz de correlaciones en el conjunto de entrenamiento estandardizado.

Como se puede observar a simple vista puede parecer que haya variables cuya correlación sea prácticamente igual a 1, por lo que podrían ser suprimibles para el proceso de entrenamiento. En concreto, si nos quedamos con el triángulo superior y buscamos los valores mayores a 0.95, obtenemos:

```
There are 23 variables which correlation with another is greater than 0.95
```

Por lo que hay 23 variables que podrían ser potencialmente eliminadas.

La decisión sobre qué características son más relevantes para el entrenamiento, una vez mostrado empíricamente que hay variables altamente correladas, la vamos a hacer utilizando el **Análisis de componentes principales** (PCA). Las *componentes principales* de un conjunto de datos son una secuencia de vectores unitarios ortogonales entre sí y que marcan las direcciones que ajustan mejor a nuestro conjunto de datos, minimizando la distancia cuadrática media desde los puntos a la recta generada por cada vector. PCA es el proceso de encontrar estas componentes principales.

Una vez se han hallado, se reduce la dimensionalidad de nuestro conjunto de datos proyectando cada punto de datos a sus direcciones principales para obtener datos de menor dimensión, pero preservando la variabilidad de los datos lo máximo posible.

Se puede probar de hecho que las componentes principales son los vectores propios de la matriz de covarianzas de nuestro conjunto de datos, por lo que para hallarlas se debe hacer la descomposición de

la matriz en valores singulares.

Tras aplicar el análisis de componentes principales, conseguimos que en nuestro conjunto de datos no haya correlaciones entre las variables. Esto nos ayuda además a reducir el *overfitting* al tener menos variables dependientes. En nuestro caso, tras aplicar PCA haciendo que el algoritmo explique el 95 % de la varianza de nuestro conjunto de datos, obtenemos que nos quedamos con 17 de las variables iniciales. Además, como podemos ver en la Figura 3, las variables están completamente incorreladas.

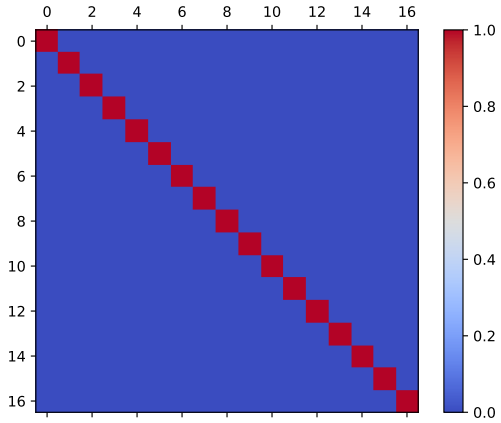

Figura 3: Matriz de correlaciones tras aplicar PCA.

1.5. Métrica de error.

En este problema, la métrica de error que utilizaremos es la estándar utilizada en problemas de regresión, el error cuadrático medio (MSE), que sabemos que viene dado por:

$$MSE(h) = \frac{1}{N}\sum_{n=1}^{N}(h(x_n) - y_n)^2.$$

Donde sabemos que N es el tamaño de la muestra usada e y_j es el valor que toma la función f en el punto x_j para cada $j = 1, \ldots, N$.

Esta métrica de error penaliza mucho los *outliers* pues la distancia entre un punto lejano y el valor que predigamos mediante la regresión será grande, y error se incrementará en gran medida. Este puede ser un motivo que nos anime a tratar de eliminar un porcentaje de datos que consideremos como *outliers*. Hay que recordar además que no está acotada superiormente, por lo que podemos obtener valores muy grandes de error.

Sin embargo, esta métrica es idónea pues para que la regresión sea buena, lo que se pretenderá es que dentro de este conjunto de datos las distancias entre el valor predicho por nuestra regresión y el valor que tenemos como dato, y_i, sean lo más parecido posibles, por lo que es sin duda la mejor métrica de error a usar.

1.6. Regularización y parámetros del modelo.

A la hora de entrenar, se demuestra tanto empírica como teóricamente que aplicar **regularización** mejora sustancialmente el resultado de los modelos. En la teoría, la regularización nos limita la clase de funciones a utilizar, reduciendo así la dimensión VC de la misma, y por tanto mejorando la cota del error fuera de la muestra que podemos dar. Además, en la práctica, la regularización previene a nuestro modelo de *sobreajustar* la muestra.

Existen muchos tipos de regularización que se pueden aplicar a la hora de entrenar. Comentaremos dos de las más frecuentes y utilizadas habitualmente.

- La regularización **Lasso** o **L1**, que también añade un érmino de penalización a la función de pérdida, pero que en este caso suma el valor absoluto de los pesos:

$$L_{reg}(w) = MSE(w) + \lambda \sum_{i=0}^{N} |w_i|.$$

- La regularización **Ridge** o **L2** suma una término cuadrático a modo de penalización a la función de pérdida. Este término es equivalente al cuadrado de la norma de los pesos. La nueva función de pérdida queda como:

$$L_{reg}(w) = MSE(w) + \lambda \|w\|_2^2.$$

  Como hemos visto en teoría, esto es idéntico a minimizar el error cuadrático medio sujeto a que $\sum_{j=0}^{p} w_j^2 < c$ para cierto $c \in \mathbb{R}$. Así, estamos haciendo que los coeficientes sean más pequeños y reduciendo la complejidad del modelo.

En ambos casos, estamos añadiendo a la pérdida una penalización multiplicada por un parámetro λ. Si reducimos la constante de penalización λ, el término que nos queda es igual que el error cuadrático medio, por lo que lo interesante sera ajustar bien este parámetro para que la regularización afecte de manera positiva al entrenamiento.

La diferencia entre ambas es que en la regularización L_1 ayuda a seleccionar variables eliminando aquellas que tienen menos relevancia. La L_2 funciona mejor cuando se piensa que todas las variables son relevantes para la predicción. Además, el término que ésta introduce es diferenciable lo cual tiene ventajas computacionales.

En este caso ya habíamos tratado de seleccionar las características que mejor explicasen la varianza del conjunto mediante PCA. Es por todo ello que elegimos usar la regularización L_2 en este problema.

Quedaría por discutir los demás hiperparámetros con los que vamos a realizar nuestro entrenamiento. Estimar los mejores parámetros para un modelo es una tarea compleja. La mejor opción en la práctica es tomar para cada hiperparámetro un conjunto de valores que sepamos empíricamente que han dado buenos resultados en problemas similares y hacer una búsqueda haciendo combinaciones de esos valores de hiperparámetros para tratar de encontrar cuál es la combinación que mejor se ajusta a nuestro problema completo. Todo ello lo podemos realizar con la función de `sklearn` : `GridSearchCV`.

Esta función, recibe como parámetros:

1. `estimator` el estimador que va a utilizar para aproximar lo que nos interese. Podemos usar SVMs, Regresores Lineales, Perceptron... En nuestro caso, usaremos
   - `SGDRegressor` que nos realiza la regresión usando el descenso de gradiente estocástico.
   - `Ridge` que nos realiza la regresión usando la regularización $L2$.
2. `param_grid`, un diccionario que especifica para cada estimador que le demos el conjunto de hiperparámetros por los que tendrá que explorar.
3. `scoring`, un string que indica cuál es la estrategia para evaluar el resultado de la validación cruzada. En nuestro caso, debemos especificar `"neg_mean_squared_error"`, pues queremos obtener el modelo que **menor** error cuadrático medio obtenga, y `GridSearchCV` siempre intentar **maximizar** la estrategia proporcionada.
4. `n_jobs`, que indica cuántos procesos correr en paralelo. Elegimos la opción -1 para que se hagan todos los posibles y acelerar así el entrenamiento.

1.7. Selección de hipótesis.

Referencias

[1] Kam Hamidieh. "A Data-Driven Statistical Model for Predicting the Critical Temperature of a Superconductor". en. En: *arXiv:1803.10260 [stat]* (oct. de 2018). arXiv: 1803.10260. URL: http://arxiv.org/abs/1803.10260 (visitado 25-05-2021).

[2] *Superconductivty Data Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data (visitado 12-10-2018).