UNIVERSIDAD DE GRANADA

AA

Aprendizaje Automático

Práctica 3: Regresión

Autores:
Mario Carmona Segovia

Profesor: Pablo Mesejo Santiago

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Curso 2020 - 2021

Índice

Índice de figuras

Índice de tablas

1. Analizar y comprender el problema a resolver

El problema de regresión que vamos a utilizar, esta relacionado con la superconductividad. Un superconductor es un material que es capaz de ofrecer cero resistencia al paso de la electricidad por él, pero esta propiedad no se da siempre; hace falta que el material se encuentra por debajo de cierta temperatura, llamada temperatura crítica. En este problema de regresión se trata de estimar la temperatura crítica de un superconductor a partir de varios características de él, que han sido extraídas de la fórmula química de los superconductores. Como se indica en el paper [1], este modelo no predice si un material es superconductor o no, sino que da predicciones sobre a partir de que temperatura pasa a ser superconductor. Según el paper [1], el radio atómico, la valencia, la afinidad electrónica y la masa atómica son las características que más contribuyen a la precisión de la predicción del modelo. Esta propiedad de los superconductores es muy útil en el mundo científico para la realización de experimentos, mientras que para la vida cotidiana no tanto debido a que las temperaturas críticas son muy bajas y difíciles de mantener.

1.1. Elementos del problema

1.1.1. Datos de entrada (X)

Los datos de entrada se componen de 21263 ejemplos de materiales superconductores, teniendo 81 características distintas cada ejemplo. Entre estas características podemos encontrar: el número medio de elementos, la masa atómica media, densidad media, etc. Dentro del fichero de datos, los datos de entrada se encuentran en las primeras 81 columnas. Todos los datos son de tipo numérico.

Dentro de estas 81 características en el paper [1] se indica que hay 8 grupos de variables principales:

Variable	**Unidades**	**Descripción**
Masa atómica	Unidades de masa atómica (AMU)	Masas totales en reposo de protones y neutrones
Primera energía de ionización	Kilo-Julios por mol (kJ/mol)	Energía requerida para eliminar un electrón de valencia
Radio atómico	Picometros (pm)	Calculo del radio atómico
Densidad	Kilogramos por metro cúbico (kg/m^3)	Densidad a temperatura y presión normales
Afinidad electrónica	Kilo-Julios por mol (kJ/mol)	Energía requerida para añadir un electrón a un átomo neutro
Temperatura de fusión	Kilo-Julios por mol (kJ/mol)	Energía necesaria para pasar de sólido a líquido sin cambiar la temperatura
Conductividad térmica	Watts por metro-Kelvin (W/(m K)	Coeficiente de conductividad térmica K
Valencia	No tiene unidades	Número típico de enlaces químicos formados por el elemento

Para conseguir las 81 características se obtienen valores de 10 métricas de cada variable.

Las métricas de las que se obtiene el valor son las siguientes:

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

Con esto se obtiene 80 características, y para completar la 81 características se añade el número de elementos.

1.1.2. Salida (Y)

La salida del problema es la temperatura crítica del superconductor. La temperatura crítica de cada ejemplo de los datos, se especifica en la columna número 82 de los datos. Esta temperatura crítica se mide en Kelvins (K).

1.1.3. Función del problema (f: X → Y)

La función del problema consistirá en a partir del valor de la distintas características de un superconductor, obtener la temperatura crítica del superconductor.

1.2. Relación entre las características

Para comprobar si los datos del problema están relacionados, ya sea positivamente o negativamente, voy a calcular la matriz de correlaciones de los atributos de todos los datos de entrada para entrenamiento.

Una correlación es un valor entre -1 y 1, que indica la relación positiva o negativa entre las variables. Cuanto más cerca del 0, menos relación tienen los atributos.

Para calcular la matriz de coeficientes de correlación se hace uso de la función corrcoef de numpy [2]. Esta función calcula la correlación de Pearson de cada par de características, y devuelve los valores como una matriz. Los argumentos más importantes de esta función son:

- x: Matriz de datos
- rowvar: Si es igual a True indica que los atributos son las distintas filas de la matriz de datos, si es igual a False son las distintas columnas de la matriz de datos

De la matriz de correlaciones calculada se puede deducir la siguiente información:

Mínimo	0.00023
Media	0.37

Tabla 1: Datos obtenidos de la matriz de coeficientes de correlación

- Como la mínima correlación no es cero, todas las variables están relacionadas.
- Como la media de correlaciones tiene un valor alto para la gran cantidad de características que tiene el problema, puede que sea posible una reducción de la dimensionalidad.

Para tener una mejor idea de la correlación que hay entre las variables, he generado la siguiente gráfica donde es más visible la alta correlación de los valores:

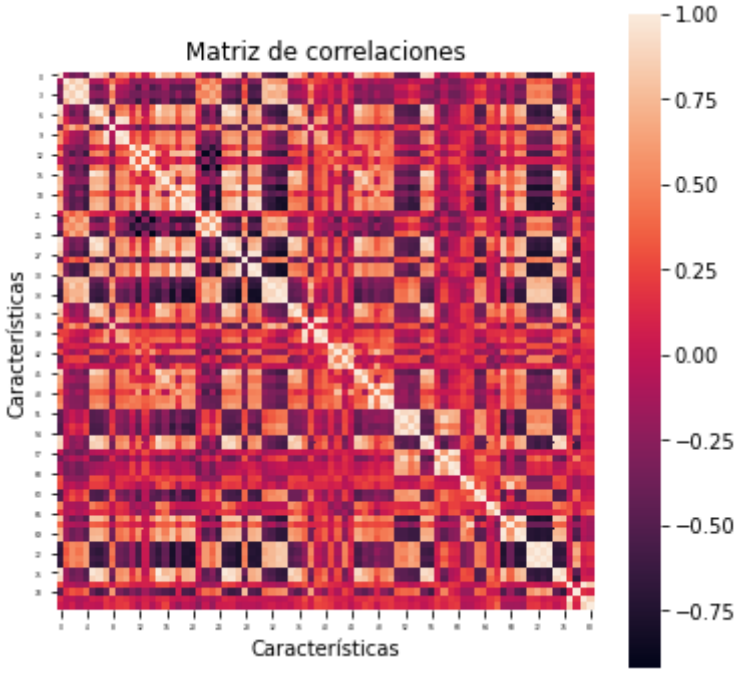

Figura 1: Matriz de correlaciones

En la figura se puede comprobar como hay mucha variabilidad en las zonas de color, y no hay grandes zonas con poca correlación.

Cuando dos variables tienen mucha correlación quiere decir que se puede eliminar una de ellas, ya que son casi equivalentes.

Dado que se ha obtenido un valor medio de correlación alto, lo más normal sería probar a reducir la dimensionalidad con alguna técnica como PCA, pero si tenemos en cuenta lo que se indica en el paper [1] no es recomendable realizar PCA, ya que según el paper no se obtiene ningún beneficio de la reducción de dimensionalidad; y según el paper esto es debido a la necesidad de una gran cantidad de características para capturar un porcentaje sustancial de la varianza.

2. Elección de la clase de funciones

La clase de funciones (H) que se va a usar, es la clase de los polinomios de orden 1; porque es una clase simple de la que se podrá ajustar un buen modelo dada la cantidad de datos de los que disponemos.

La función a ajustar será una combinación lineal de los todas las características con sus respectivos pesos.

$$\hat{y}(w, x) = w_0 + \sum_{n=0}^{N} w_n x_n$$

Siendo N el número total de características y los w's los coeficientes del modelo.

He decidido no realizar transformaciones no lineales porque no he encontrado en la literatura ningún sugerencia que haya dado buenos resultado, ni tengo el suficiente conocimiento sobre el campo del problema como para encontrar alguna relación entre las características.

3. Identificar el conjunto de hipótesis

Dado que el problema a resolver es un problema de regresión, los algoritmos a usar para ajustar la función son los siguientes:

- Ridge
- Stochastic Gradient Descent (SGD)

3.1. Ridge

He decidido usar Ridge porque es un método muy rápido que puede dar buenos resultados.

El método de Ridge es equivalente a realizar el método de la pseudo-inversa junto con la regularización L2.

$$w = X^{'} y$$
$$X^{'} = (X^T X + \lambda I)^{-1} X^T$$

Los parámetros más importantes de la función del método Ridge en scikit learn [3] son los siguientes:

- alpha: representa la intensidad de la regularización
- normalize: indica si se deben normalizar los datos o no. En mi caso lo dejo por defecto porque previamente al ajuste se realizar un preprocesado de los datos.
- max_iter: número máximo de iteraciones
- solver: solver para calcular los coeficientes del modelo

Esta función de scikit learn usa siempre como función de error el error cuadrático medio, explicado en el apartado [6], y usa siempre una regularización L2.

3.2. SGD

He decidido usar SGD porque es una técnica de optimización que lleva mucho tiempo usándose en el machine learning. Como se indica en la gría de scikit learn [4], SGD no es ningún modelo de aprendizaje automático, sólo es una técnica de optimización. Este algoritmo da buenos resultados con conjuntos de datos grandes, como puede ser nuestro problema que tiene más de 10.000 ejemplos. Según los argumentos que se utilicen para esta técnica se puede llegar a parecer a otras técnicas de optimización, como regresión logística o Ridge.

SGD es una variante del algoritmo de gradiente descendente. Esta variante intenta solucionar el problema de caer en mínimo o máximo locales, intentando separar en varios partes los datos

que guían la búsqueda, para que el avance del algoritmo no se vea afectado por unos pocos datos que son malos para el ajuste.

$$h(x) = \sigma(w^T x)$$

Los parámetros más importantes de la función del método SGD para regresión en scikit learn [5] son los siguientes:

- loss: Función de pérdida a usar
- penalty: Tipo de regularización
- alpha: representa la intensidad de la regularización
- max_iter: número máximo de iteraciones
- learning_rate: forma de calcula la tasa de aprendizaje en cada iteración
- eta0: tasa de aprendizaje inicial

Una desventaja de SGD es que es sensible a la escala de las características, por lo que es imprescindible realizar una normalización para obtener buenos modelos con esta técnica.

4. Separación de los datos en training y test

Dado que disponemos de un conjunto de datos, que no están separados previamente en training y test, debemos realizar un proceso de validación, separando por nuestra cuenta los datos. Para este problema se ha decidido separar un 20 % de los datos para test y el resto para training. Como no hay criterio fijo en la elección del tamaño de ambos conjuntos de datos, me he decidido por la regla general que se sigue, que es dividir entorno al 20 % de los datos para test. Aunque también podría haber usado un tercio de los datos para test, dado que en el paper [1] se indica que ellos han realizado la validación con un 30 % de los datos y el modelo ha dado buenos resultados de predicción.

Una vez decidida la proporción de datos para cada parte, toca elegir un método para dividirlos de forma que los datos sigan siendo independientes e idénticamente distribuidos. Dado que se trata de datos para un problema de regresión, el método seguido a sido una simple separación de los datos, de forma que el 80 % inicial del conjunto de datos ha sido usado para training y el restante 20 % para test ó conjunto de validación.

Una vez realizada la separación en training y test, los datos de test no se deben tocar durante todo el proceso de elección de la hipótesis final, ya que sino se realizaría data snooping, es decir, se está contaminando el ajuste del modelo con los datos de test, por lo que el modelo resultante dará buenos resultados con el test, pero no será un modelo correcta para fuera de la muestra, ya que al sufrir data snooping sea perdido generalidad.

	Tamaño	**Porcentaje**
X	21263	100 %
X_train	17011	80 %
X_test	4253	20 %

Tabla 2: Separación de los datos de entrada

	Tamaño	**Porcentaje**
Y	21263	100 %
Y_train	17011	80 %
Y_test	4253	20 %

Tabla 3: Separación de los datos de salida

5. Preprocesado de los datos

Para que el entrenamiento del modelo sea lo más eficaz y eficiente posible, se deben preprocesar los datos de training.

El preprocesamiento de los datos usado consiste en los siguientes pasos:

- Eliminación de los datos sin variabilidad
- Reducción/Aumento de la dimensionalidad
- Normalización de los datos
- Codificación de los datos
- Datos faltantes
- Datos extremos

5.1. Eliminación de los datos sin variabilidad

Se decide eliminar las características sin variabilidad porque una característica sin variabilidad no aporta nada al aprendizaje, por lo que se eliminan esas características para no malgastar recursos. Una característica sin variabilidad es aquella en la que no varía su valor en los distintos ejemplos de los datos de entrada.

En los datos de este problema no hay datos sin variabilidad.

5.2. Reducción/Aumento de la dimensionalidad

Para comprobar la distribución de varianza explicada entre las características he decidido usar PCA. PCA es una técnica para la reducción de dimensionalidad de los datos mediante la descomposición de valores singulares.

Como PCA es una técnica muy sensible a la diferencia de escalas en las características, se deben normalizar los datos antes de usar PCA.

Una vez realizada la técnica PCA sobre los datos, se obtienen las varianzas que explica cada característica.

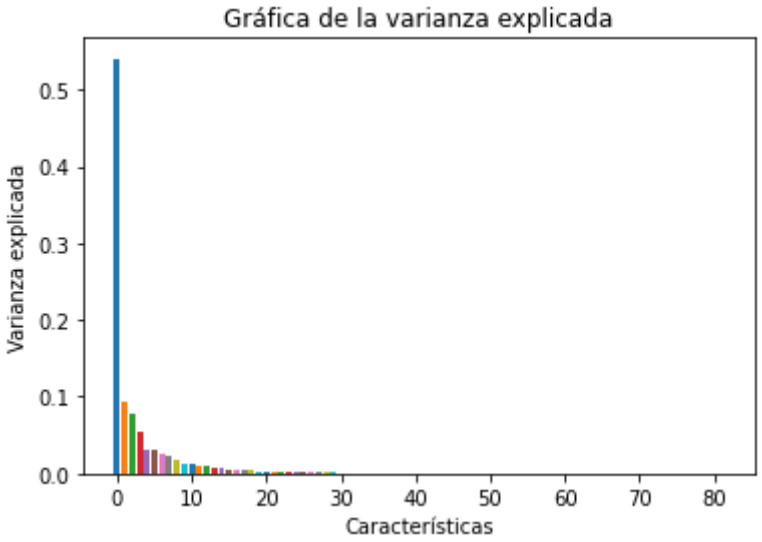

Figura 2: Varianza de cada característica

Como se puede ver en esta figura, la mayoría de las varianza está concentrada en unas pocas variables.

Para comprobar con cuantas variables se podría explicar al menos un 99 % de la varianza he calculado la varianza acumulada de cada característica, y he obtenido la siguiente gráfica:

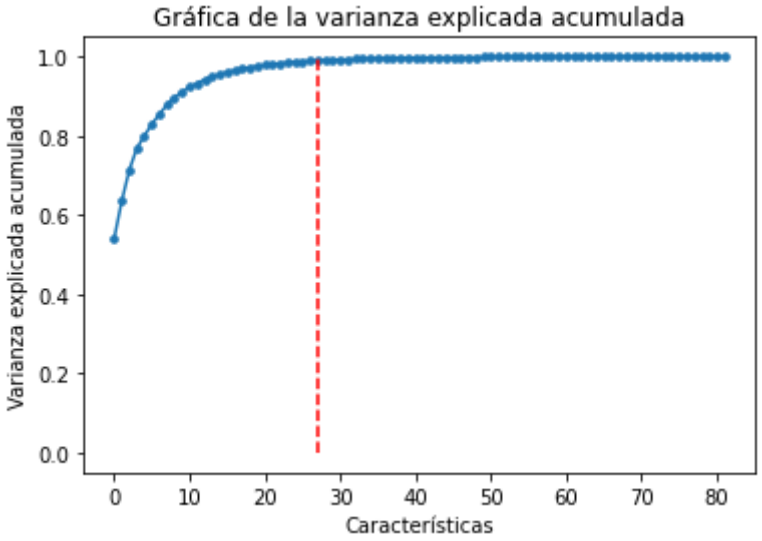

Figura 3: Varianza acumulada de cada característica

En esta figura se indica con una línea roja la primera característica que tiene al menos un 99 % de la varianza explicada. Según la gráfica, con 27 características se puede explicar al menos el 99 %. Por lo que en principio con 27 características se obtendría casi los mismos resultados

que con todas las características.

Como se indica en el apartado [1]. He optado por no realizar ninguna reducción o aumento de la dimensionalidad.

5.3. Normalización de los datos

He decidido normalizar los datos de las distintas características porque tienen escalas y rangos muy distintos. Esto se puede comprobar con la siguiente gráfica donde se muestra el mínimo, máximo y media de cada característica.

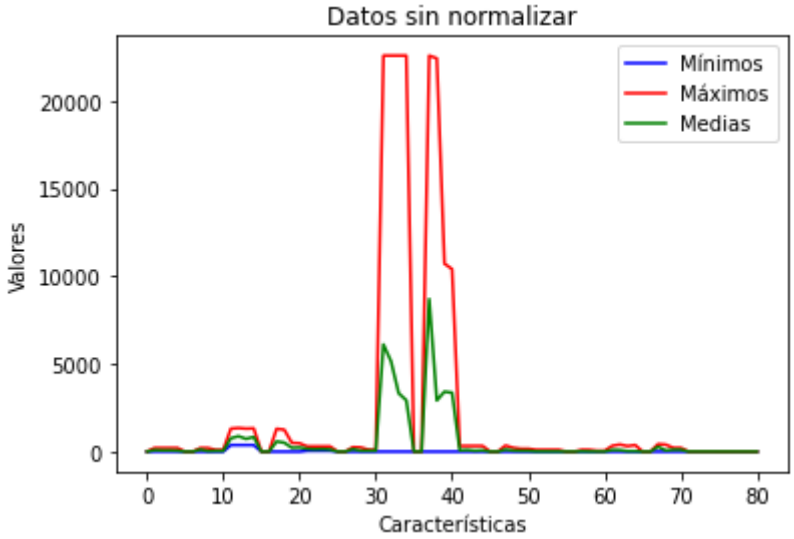

Figura 4: Datos sin normalizar

Para evitar que esa diferencia de escalas influya en el aprendizaje, como le pasa a SGD o a la regularización; más que la propia influencia de la característica en el aprendizaje, se deben normalizar los datos. Se han normalizado los datos en un rango de valores [0,1], para ello se ha utilizado la función MinMaxScaler de la librería scikit learn [6].

Esta función consiste en escalar y trasladar cada ejemplo individualmente al rango que se le ha pasado como parámetro, en mi caso el rango es [0,1].

$$X\_desviEstantar = (X - X\_min)/(X\_max - X\_min)$$
$$X\_escala = X\_desviEstandar * (max\_rango - min\_rango) + min\_rango$$

El principal parámetro de la función es feature_range, que fija el rango de valores que acota el valor de los datos.

Una vez realizada la normalización de los datos de entrenamiento, en la siguiente gráfica se puede ver el resultado de esta.

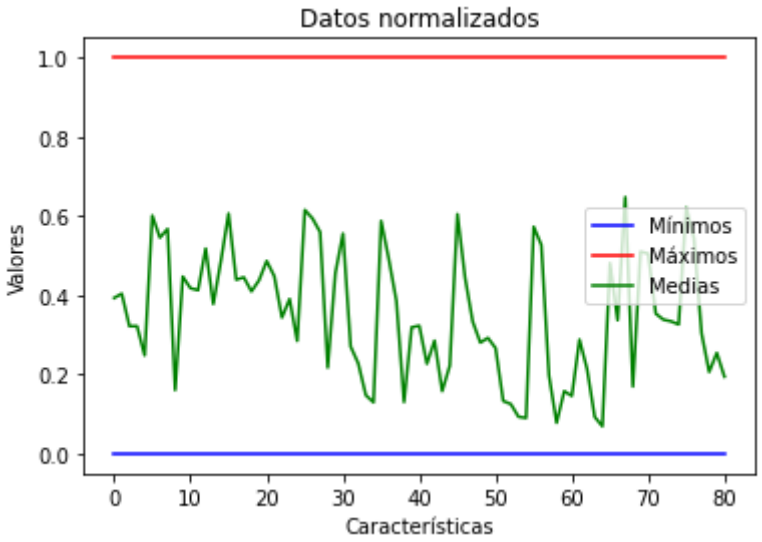

Figura 5: Datos normalizados

Como se puede apreciar en la figura [5], ahora los datos están acotados en el rango [0,1]. Ahora los rangos de las características no afectarán tanto y mejorará el aprendizaje.

5.4. Codificación de los datos

No hace falta codificar los datos, ya que si comprobamos los datos de entrada se puede ver que todas las características tiene un valor numérico, por lo que se dejan tal cuál están los datos.

5.5. Datos faltantes

En este apartado tampoco hace falta procesar los datos, ya que como se puede comprobar en la descripción de los datos en su repositorio UCI [7], se indica que no hay datos faltantes en los datos.

5.6. Datos extremos

La eliminación de los datos extremos es una operación delicada, porque podemos llegar a eliminar datos que no son extremos y que son útiles para conseguir unos modelos de mayor calidad, que tienen en cuenta una mayor cantidad de situaciones.

Para eliminar los outliers he usado la función LocalOutlierFactor de la librería scikit learn. Esta función calcula la densidad local de cada ejemplo, respecto a un cierto número de vecinos. Cuanto mayor se cree que va a ser el número de outliers, se debe aumentar el número de vecinos. En la guía de scikit learn [8] se indica que si no se sabe con cierta certeza el grado de outliers que hay en los datos, un valor adecuado para el número de vecino es 20, que es el valor por defecto; con este valor se trabajará bien en general. Como yo no tengo claro el grado de outliers,

he utilizado ese valor, y además no se ha indicado la contaminación. La métrica para la distancia sea dejado por defecto porque no he encontrado ninguna ventaja en cambiar la métrica.

Los principales argumentos de esta función son:

- n_neighbors: número de vecinos a tener en cuenta en el calculo de la densidad local
- metric: es la métrica con la que se mide la distancia entre el ejemplo y los vecinos (distancia manhattan, distancia euclídea, etc)
- contamination: es el grado de outliers que se espera que haya en los datos

Esta función devuelve un vector de enteros negativos, con un tamaño igual al número de ejemplos; que representa la densidad local del ejemplo. Si el valor es mayor o igual que -1, se considera que el ejemplo es un inlier. Si el valor es menor que -1 se considera que el ejemplo es un outlier.

Para reducir el riesgo de eliminar ejemplos importantes, he decidido eliminar los ejemplos que tenga un valor de densidad menor que el valor del tercer cuartil más 10 veces la distancia intercuartílica.

Con este criterio se han eliminado 842 ejemplos (4.95 %).

6. Justificación de la métrica de error

Dado que se trata de un problema de regresión, he decidido no coger métricas que son muy usadas en clasificación como accuracy, specificity, etc. De entre las métricas de error más usadas, he decido escoger el error cuadrático medio (MSE), ya que es una métrica que hemos usado mucho en teoría y en anteriores prácticas, y además es la métrica de error usada en el paper [1] para el modelo XGBoost.

El cálculo del MSE se realiza con la siguiente fórmula:

$$MSE = (w^T x - y)^2$$

A pesar de que se use MSE como función de error en los modelos, se usará la raíz del error cuadrático medio (RMSE) para valorar los modelos, ya que hace que sean más fáciles de visualizar y comparar los resultados.

También se podría usar RMSE como función de error, ya que es un métrica muy común y está considera como una excelente métrica para predicciones numéricas, tal y como se indica en el paper [9].

Para obtener el valor del RMSE sólo hace falta hacer la raíz cuadrada del valor del MSE.

$$RMSE = \sqrt{MSE}$$

7. Justificación de los hiperparámetros

7.1. Modelos

7.1.1. Ridge

En este método hay que fijar los hiperparámetros siguientes:

Iteraciones máximas

Este parámetro indica el número máximo de iteraciones que realiza el método Ridge. He decidido dejar el valor por defecto, 1000, ya que no he encontrado ninguna razón por la que debería cambiarlo.

Solver

Como solver he elegido Singular Value Descomposition (SVD), ya que es necesario para usar como función de pérdida el error cuadrático medio y realizar el método de la pseusoinversa.

En el método SVD, los datos de entrada, X, representados como una matriz, se pueden descomponer como producto de tres matrices, el producto de UDV^T. U y V son matrices ortogonales, y D es una matriz diagonal, es decir, sólo tiene valores en la diagonal principal. Las dimensiones de estas tres matrices dependen de las dimensiones de X. U tiene que tener el mismo número de filas que X; y rl número de columnas de V^T debe ser igual al número de columnas de X. D representa los valores singulares, la parte más importante de esta descomposición.

Los valores singulares nos dan el rango de la matriz, es decir, nos dicen si tiene determinante o no y por consecuente si tiene inversa o no. El número de valores singuales distintos de 0 son el rango máximo de X.

A parte se debe fijar el alpha para la regularización, pero este hiperparámetro se explica en la sección de regularización [7.2].

7.1.2. SGD

En este método hay que fijar los hiperparámetros siguientes:

Función de pérdida

Como función de pérdida he elegido el error cuadrático medio, tal y como se explica en el apartado [6].

Penalty

Este parámetro indica que tipo de regularización se va a realizar junto con la técnica SGD. Hay tres opciones: L1, L2 ó Elactic net.

En mi caso sólo he probado con L1 y L2, ya que son regularizaciones más sencillas y que además se han visto en teoría.

Ambas regularizaciones son explicadas en el apartado [7.2].

Tasa de aprendizaje (eta)

Como forma de calcular la tasa de aprendizaje en cada iteración, he elegido que se mantenga constante, por lo que el valor indicado en el eta inicial, se mantendrá durante toda la ejecución del método.

Tasa de aprendizaje inicial (eta0)

Este parámetro indica el tamaño de los descensos del gradiente. Cuanto mayor sea su valor, mayores son los descensos en cada iteración.

A priori el valor más óptimo del eta inicial para este problema se desconoce, pero en el paper [1] se indican varios valores de eta inicial, con los que en el paper se han obtenido buenos resultados, por lo que he decidido usar los mismos valores. Estos valores son: 0.01, 0.015, 0.02 .

Iteraciones máximas

Este parámetro indica el número máximo de iteraciones que realiza la técnica SGD. Para calcular este valor máximo se usa la siguiente fórmula:

$$max\_iter = np.ceil(\frac{10^6}{n}) * 2$$

Siendo n el número de ejemplo del problema.

Esta fórmula la he tomado de la guía de SGD [4] con la única modificación de multiplicar por dos. Hago esta multiplicación porque con la fórmula por defecto que aparece en la guía me daba warnings con algunas hipótesis, por lo que he decidido aumentar el límite máximo un poco. En el caso de este problema es igual a 124.

A parte se debe fijar el alpha para la regularización, pero este hiperparámetro se explica en la sección de regularización [7.2].

7.2. Regularización

La regularización es un método para evitar realizar overfitting (sobreajuste), es decir, entrenar demasiado con el modelo. Para evitar esta situación, la regularización aplica una penalización a los pesos del modelo. Esta penalización cambia en cada tipo de regularización.

Siempre es bueno realizar regularización sobre un modelo, ya que no afecta en gran medida si no mejora el modelo, y nos proporciona una gran reducción de varianza a cambio de un pequeño incremento en el sesgo.

La regularización L1 o Lasso penaliza la suma del valor absoluto de los coeficientes de regresión. Esta penalización fuerza a que los coeficientes de los predictores tiendan a cero. Cuando un predictor tiene un coeficiente de regresión igual a cero no influye en el modelo, de esta forma esta regularización excluye del modelo a los predictores menos relevantes.

La regularización L2 o Ridge penaliza la suma de los coeficientes elevados al cuadrado. Esta penalización fuerza a reducir de forma proporcional el valor de todos los coeficientes del modelo pero sin que estos lleguen a cero. Que no lleguen a cero los coeficientes es la mayor diferencia con la regularización L1.

La intensidad de ambos tipos de regularización está contralada por el hiperparámetro alpha (λ). Cuando alpha vale cero equivale a no realizar regularización en el modelo. A medida que alpha aumenta, mayor es la penalización y más predictores quedan excluidos.

A priori no se sabe cual será el valor de alpha que nos dará mejores resultados, por ello he ido probando valores comenzando en 1 e ir dividiendo entre 10, con el resto de hiperparámetros fijados como se indica en el apartado de cada algoritmo.

Los valores escogidos para el algoritmo Ridge son: 0.1, 0.01 . Esto se ha decidido comprobando los resultados con distintos valores alpha, los cuáles se pueden ver en la siguiente tabla:

Apha	**Accuracy**
1	15.38
0.1	16.01
0.01	16.22

Tabla 4: Pruebas con alpha - Ridge

Los valores escogidos para el algoritmo SGD son: 0.001, 0.0001 . Esto se ha decidido comprobando los resultados con distintos valores alpha, los cuáles se pueden ver en la siguiente tabla:

Apha	**Accuracy**
1	18.31
0.1	16.53
0.01	16.31
0.001	16.04
0.0001	16.88

Tabla 5: Pruebas con alpha - SGD

8. Selección de la mejor hipótesis

Para elegir la mejor hipótesis de forma que la elección no sea optimista he usado la técnica de cross-validation. Esta técnica consiste en dividir los datos de entrenamiento en k partes del mismo tamaño, y realizar k veces el ajuste del modelo teniendo todos los fold como datos de training menos uno, y en cada iteración cambia el fold usado como test. En la práctica se realiza el k-fold cross-validation, en vez de usar la técnica Leave One Out, que consiste en dividir todos los datos en grupos de un ejemplo, esta técnica en la práctica es computacionalmente imposible para elegir una hipótesis ya que habría que ajustar tantos modelos como ejemplos hay en los datos.

Las ventajas de usar esta técnica es que nos aseguramos de que no vamos a elegir una hipótesis tomando valores de error optimistas, sino al contrario. Esta técnica nos permite eliminar gran parte de la influencia de los datos usados para entrenar sobre el modelo, para obtener un modelo con más generalidad y cuyo error será tomado de forma pesimista.

De cada hipótesis se obtiene la media del error de cross-validation en cada iteración de la técnica.

La mejor hipótesis será la que obtenga un menor error de cross-validation.

Las distintas hipótesis serán los distintos modelos con todas las combinaciones de hiperparámetros definidos en el apartado [7].

Para realizar k-fold cross-validation se ha usado la función GridSearchCV de la librería scikit learn [10]. Esta función tiene los siguientes parámetros importantes:

- estimator: indica la técnica a usar para ajustar los modelos
- param_grid: lista que indica para los distintos valores de los parámetros del estimador. Los valores de los parámetros que no se indican en esta lista se dejan por defecto.
- cv: número que indica el número de folds a usar en la validación cruzada
- error_score: métrica de error para valorar las hipótesis
- return_train_score: valor booleano que indica si se devuelve o no el error de training de los modelos ajustados en la validación cruzada. Si se pone True, se devuelve este error, pero hay que tener en cuenta que esto llevará un sobrecoste en el tiempo que necesitará la técnica. Si se pone False seguirá devolviendo el mejor modelo sin ningún problema y tardará menos, pero no dispondremos de la información de training.

Como estimador yo he elegido los métodos indicados en el apartado [3].

Como parámetros se han puesto los valores de los hiperparámetros indicados en el apartado [7] para cada método.

Se ha realizado una validación cruzada con 10 folds, por lo que he realizado una 10-fold cross-validation, ya que cuanto mayor sea el número, mayor generalidad se obtiene, pero mayor coste en tiempo conlleva ya que hay que realizar más ajustes; pero como el número de hipótesis no es demasiado elevado y una de las técnicas es muy rápida, Ridge; he decidido que es más

conveniente que realizar por ejemplo una 5-fold cross-validation. Lo más general es usar 5 o 10 folds.

Como métrica de error para evaluar un modelo se ha usado RMSE negativa, por lo que habría que cambiar de signo del error para que sea igual al RMSE tal y como se indica en el apartado [6]. El error de validación cruzada se aproxima al E_out de la hipótesis.

Y se indica que no se devuelvan los errores obtenidos para training, para no malgastar recursos en obtener datos que no voy a usar.

Para cada método he obtenido el E_cv de todas las hipótesis de cada método.

Para el método Ridge, la mejor hipótesis según el error de validación cruzada es el siguiente:

Alpha	0.1
E_cv	18.34

Tabla 6: Mejor hipótesis del método Ridge

Para el método de SGD, la mejor hipótesis según el error de validación cruzado es la siguiente:

Penalty	L1
Alpha	0.0001
Eta	0.01
N_iter	74
E_cv	18.99

Tabla 7: Mejor hipótesis del método SGD

Viendo la mejor hipótesis de ambos métodos, nos quedamos con la mejor hipótesis del método Ridge, que es la que menor error tiene.

9. Cálculo del error de la hipótesis final

Una vez escogida la hipótesis final, ahora debo volver a entrenar el modelo elegido pero con todos los datos de training, sin realizar la validación cruzada.

Antes de obtener las predicciones de los datos de test, se deben preprocesar los datos de entrada de test, pero utilizando los datos de training para ello.

Una vez preprocesados los datos, obtengo las predicciones de los datos de test, y obtengo el error RMSE en base a las predicciones y a las salidas verdaderas de test.

El error de test obtenido es 16.01 . Este E_test se aproxima al E_out de la hipótesis final.

Para poder comparar este error y ver si es bueno o no, he calculado el error de un modelo que predice para todo ejemplo la media de las salidas de los datos.

El E_test obtenido para este predictor de la media es 23.33, este error se podría considerar como el máximo error que se debería obtener.

Dado que el error del modelo ajustado es mucho mejor que el error del predictor medio considero que es un buen ajuste.

Además se puede comprobar que es un buen resultado porque en el paper [1] se indica por ejemplo que en el modelo de regresión se obtiene sobre un 17.6 de RMSE.

Referencias

[1] Kam Hamidieh. "A data-driven statistical model for predicting the critical temperature of a superconductor". En: *Computational Materials Science* 154 (2018), págs. 346-354. ISSN: 0927-0256. DOI: https://doi.org/10.1016/j.commatsci.2018.07.052. URL: https://www.sciencedirect.com/science/article/pii/S0927025618304877.

[2] *Manual de Numpy: Corrcoef*. URL: https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.

[3] *Manual de Scikit-learn: Ridge*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html.

[4] *Guia de Scikit-learn: SGD*. URL: https://scikit-learn.org/stable/modules/sgd.html#sgd.

[5] *Manual de Scikit-learn: SGD Regressor*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html.

[6] *Manual de Scikit-learn: MinMaxScaler*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html.

[7] *UCI: Superconductivity Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data.

[8] *Guia de Scikit-learn: Local Outlier Factor*. URL: https://scikit-learn.org/stable/modules/outlier_detection.html.

[9] Simon P. Neill y M. Reza Hashemi. "Chapter 8 - Ocean Modelling for Resource Characterization". En: *Fundamentals of Ocean Renewable Energy*. Ed. por Simon P. Neill y M. Reza Hashemi. E-Business Solutions. Academic Press, 2018, págs. 193-235. ISBN: 978-0-12-810448-4. DOI: https://doi.org/10.1016/B978-0-12-810448-4.00008-2. URL: https://www.sciencedirect.com/science/article/pii/B9780128104484000082.

[10] *Manual de Scikit-learn: GridSearchCV*. URL: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html.