Práctica 3

Ajuste de datos usando modelos lineales

3 de junio 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, [7], contiene características de ciertos elementos superconductores. Junto con estas características, se nos presenta una *temperatura crítica*, que en [2] 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 entrada será

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

que luego dividiremos en subconjuntos de entrenamiento y test.

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:

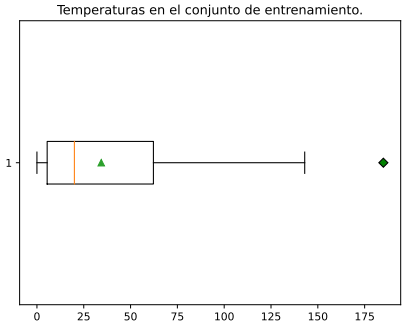

Figura 1: Diagrama de caja de las temperaturas T_c en el conjunto de entrenamiento.

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 $\mathcal{H}$.

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:

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

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.

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, se usarán particiones **estratificadas** de los datos. Esto quiere decir que , dado un número de particiones (*folds*) k, se divide el conjunto de entrenamiento en esas k particiones 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.

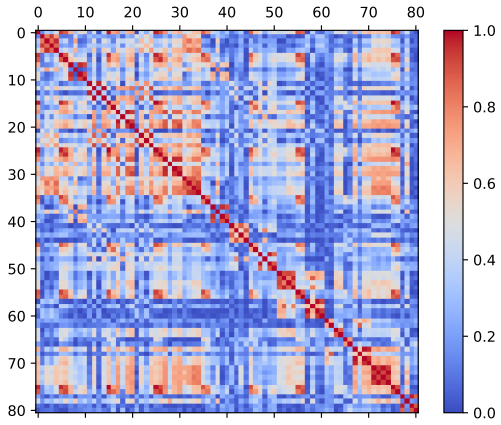

Figura 2: Matriz de correlaciones en el conjunto de entrenamiento estandardizado.

Como se puede observar a simple vista puede parecer que haya variables cuya correlación sea prácticamente igual a 1, por lo que **puede darse el caso** de que sean suprimibles en 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.

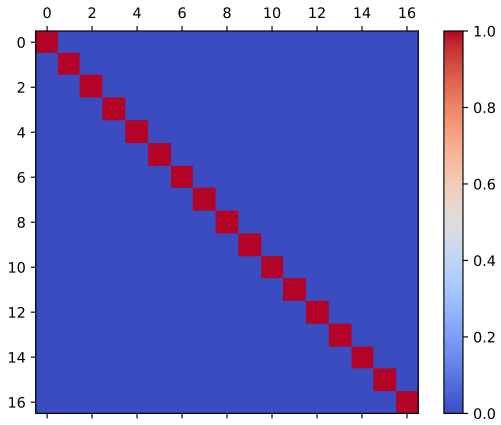

Figura 3: Matriz de correlaciones tras aplicar PCA.

Sin embargo, quedarnos con tan pocas variables no nos asegura que el modelo entrenado con estas variables vaya ser mejor que el modelo entrenado con todas las variables. Es por ello que cuando entrenemos, tomaremos para entrenar modelos con los datos reducidos en dimensionalidad y sin reducir.

Además, hay que comentar que se ha tratado de detectar **outliers** en nuestro conjunto usando `IsolationForest` de `sklearn`, pero el número de outliers encontrado era despreciable así que no se han eliminado del conjunto de entrenamiento.

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. En este caso sin embargo, no tenemos esos 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.

Hay que remarcar que para evaluar los modelos en el entrenamiento, este error se medirá en cada una

de las particiones de la validación crucada y luego se hará una media de ellos para obtener:

$$E_{cv} = \sum_{i=1}^{n} MSE_i(h),$$

que será el que se usará para determinar el modelo que escogeremos como el mejor finalmente.

Para medir el error final en el conjunto de test, usaremos también el **coeficiente de determinación** R^2. Este coeficiente mide la proporción de la varianza en la variable independiente que es predecible mediante la(s) variable(s) independiente(s). Concretamente, si consideramos $\bar{y}$ como el valor medio de las etiquetas, y consideramos las cantidades:

$$SS_{tot} = \sum_i (y_i - \bar{y})^2 \quad y \quad SS_{res} = \sum_i (y_i - h(x_i))^2,$$

calculamos R^2 como:

$$R^2 = 1 - \frac{SS_{res}}{SS_{tot}}.$$

Para interpretar esta medida, hay que saber que cuanto mejor sea el ajuste, menor es el cociente ,y más cercano es el valor de R^2 a 1, por lo que lo ideal es que R^2 sea lo más grande posible.

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. Como el número de parámetros a entrenar no es muy elevado, se ha decidido optar también por ampliar nuestras opciones en la búsqueda del mejor modelo y usar **también** esta regularización sobre el total de las características.

1.6.a. Parámetros de búsqueda.

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. Explicaremos más adelante qué conjunto de parámetros a probar le pasaremos a la función.
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.
5. `cv`, que determina el número de particiones que se hacen para la validación cruzada. En este caso, le indicamos que haga $5-$fold cross validation.

Queda por concretar los valores concretos que se le dan a los parámetros que se le pasan en el diccionario `param_grid`. En concreto, tenemos que comentar:

- El parámetro de regularización λ, que le hemos dado los valores $[0.1, 0.01, 0.001, 0.0001, 0.00001]$. Se han escogido estos valores pues se usan habitualmente en la literatura sobre los valores a escoger para el parámetro de regularización. Por ejemplo en [4].
- El número de iteraciones `max_iter`, que se le han dado los valores $[5000, 10000]$. En algunos casos, usando las 5000 iteraciones se nos indica mediante un *warning* que no se llega a converger, pero no es un problema porque tenemos el mismo modelo pero con 10000 iteraciones.
- Para el parámetro `learning_rate` usado en el regresor lineal con SGD, se utilizan también varias versiones de modificación del parámetro. En concreto, se usa `constant`, para manterner el parámetro constante, `adaptive` para que el parámetro aumente o disminuya según se vaya aumentando o disminuyendo el error obtenido, y `optimal`, para tratar de obtener el óptimo descenso en cada iteración.

1.7. Selección de hipótesis.

Llegados a este punto, es el momento de explorar nuestro espacio de parámetros para encontrar el modelo que menor E_{cv} tenga. Hay que mencionar que estos resultados se han obtenido utilizando el procesador de mi ordenador portátil: *AMD Ryzen 7 4800h with radeon graphics × 16*.

Finalmente, se decide por aplicar los mismos modelos con los mismos parámetros de búsqueda en dos preprocesados de los datos diferentes, para comprobar si la aplicación de la reducción de la dimensionalidad en este problema nos ayuda o empeora nuestros resultados. Para ello, nos basta crear dos `Pipelines` de python y ejecutar nuestro `GridScoreCV` dos veces.

Listing 1: Pipeline Standardization

```
preprocess = [
  ("standardize",StandardScaler())
]
```

Listing 2: Pipeline PCA

```
preprocess_pca = [
  ("pre-standardize",
      StandardScaler()),
  ("PCA", PCA(n_components = 0.95)
      ),
  ("standardize",StandardScaler())
]
```

Tras tener estos *pipelines* creados y nuestro espacio de parámetros para la búsqueda creados, ejecutamos el método de búsqueda.

Listing 3: Estandarizacion

```
Mejor en solo estandarizacion
----- Mejor regresor lineal
    encontrado ------
- Parametros:
Ridge(alpha=0.1, max_iter=5000)
- Error en Cross Validation
310.2545634883221
```

Listing 4: PCA

```
Mejor usando PCA
----- Mejor regresor lineal
    encontrado ------
- Parametros:
Ridge(alpha=0.1, max_iter=5000)
- Error en Cross Validation
467.96851110824326
```

Comparando los dos casos que hemos planteado para la búsqueda del mejor modelo, vemos que la hipótesis que mejor resultados nos ofrece en cuanto a minimización del E_{cv}, con una diferencia de más de 150 unidades, es el modelo que **solamente realiza estandarización** en los datos y utiliza como regresor lineal el modelo `Ridge` de `sklearn`. Es por ello que nuestra hipótesis final g será:

$$g = \text{Ridge}(alpha = 0.1, max\_iter = 5000).$$

Hay que destacar en los resultados que , cuando se utiliza preprocesamiento de los datos usando también PCA, el modelo que mejor resultados obtiene es el mismo. Esto nos puede indicar que, aunque estemos intentando mantener una gran cantidad de información explicada reduciendo las características, reducir las características nos está haciendo perder información sobre cómo se relacionan estas con la función f que tenemos que aproximar.

La regresión usando `Ridge` ha dado los mejores resultados aún variando un poco los parámetros. Se observa que para cualquier valor que se le proporcione de constante de regularización λ (alpha según `sklearn`), se obtienen errores en validación cruzada muy similares.

Además, se ha optado por usar este método de regresión porque es el que menor E_{cv} nos da, pero usando el modelo de sklearn `SGDRegressor` (que sabemos que hace regresión lineal usando SGD) con

parámetros $\lambda = 0.0001$, $max\_iter = 5000$ y *learning rate* adaptativo, se ha obtenido un $E_{cv} = 312.1983$, que se encuentra bastante cercano al error que nos da el mejor método.

Más información sobre los valores obtenidos para cada conjunto de parámetros fijo en cada uno de los algoritmos de minimización del error se puede observar en el apéndice 3.1 que se ha incluido para evitar rellenar el documento con tablas.

Mirando las tablas, podemos observar que en general el regresor que aplica la regularización de forma directa (`Ridge`), ha obtenido resultados muchos mejores mientras que `SGDRegressor` le ocurre que en algunos valores de λ y formas de actualizar η, obtiene errores en cross validation demasiado grandes. Esto es posiblemente porque necesite un número de iteraciones muchísimo más alto para encontrar una recta de regresión que aproxime nuestros datos al nivel que lo hacen los demás aproximadores. Además, con algunos valores de estos mismos parámetros, el algoritmo no es capaz hacer qeu el error se aproxime al mínimo que sabemos que podemos obtener con otros parámetros.

1.8. Error final fuera de la muestra.

Nos queda por ver cómo de bien hemos conseguido *"generalizar"* , usando el conjunto que hemos dejado desde el principio fuera de todas nuestras operaciones para evitar el *data snooping*.

Para obtener el error fuera de la muestra final, el proceso realizado ha sido simplemente ajustar primero el `pipeline` (que hace primero estandarización y luego utiliza `Ridge` como método de regresión lineal) usando los datos de entrenamiento usando la función `fit`, y a continuación predecir sobre el conjunto de test usando la función `predict` sobre el modelo ajustado.

Se calculan entonces el error cuadrático medio en el conjunto de test y el coeficiente de determinación R^2 y obtenemos lo siguiente.

```
------ RESULTADOS FINALES EN TEST -------

      - MSE: 313.4691035257312
      - R^2: 0.7366585080400442
```

2. Clasificación

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

De nuevo, comenzamos observando la información que se nos proporciona del conjunto de datos. En este caso, nuestro nuevo conjunto de datos [8] contiene características sobre impulsos eléctros que se han producido en motores. Estos motores tienen componentes intactos y componentes que están dañados. Las características se han medido en numerosas ocasiones mediante 12 condiciones de trabajo diferentes, es decir: diferentes velocidades o cargas sobre el motor. Han sido medidas usandouna sonda de corriente y un osciloscopio.

Una vez medidas las características, se han generado más datos usando la descomposición EMD [3], y se han calculado datos estadísticos como la media, desviación típica o la curtosis de las variables.

Una vez medidas las características, se determina una clase de motor a las que estas características pertenecen. Se ha dividido el conjunto en 11 clases diferentes, que nosotros trataremos de separar utilizando modelos lineales. Veamos de nuevo una tabla con más información sobre los datos:

Características	Multivariable	Número de instancias	58509
Tipo de características	Reales	Número de atributos	49
Tareas asociadas	Clasificación	Valores perdidos	$N\backslash A$

Tabla 2: Datos contenidos en el conjunto de datos Sensorless Drive Diagnosis.

Vemos sin embargo que dentro de los datos, el último valor no es una característica sino la clasificación del objeto, por lo que en realidad el número de atributos es 48.Podemos comprobar que tenemos un número de instancias aún mayor que en el caso anterior y son de nuevo variables reales. Además, tampoco tenemos valores perdidos por lo que no tendremos que preocuparnos de tratar ese caso.

Con esta información, podemos concluir que:

1. El conjunto de datos de entrenamiento serán vectores de $\mathbb{R}^{48}$ de características de un motor, por lo que podemos definirlo formalmente como:
$$\mathcal{X} = \{x_i \in \mathbb{R}^{48}, \text{ con } i = 1, \ldots, 58509\},$$
que dividiremos también en conjunto de entrenamiento y test.
2. Sabemos que los datos se han dividido en 11 clases. Además, estas clases se representan por un número del 1 al 11. Por tanto, el conjunto de llegada es:
$$\mathcal{Y} = \{y_i \in \{1, 2, \ldots, 11\} \text{ con } i = 1, \ldots, 58509\}\}$$
3. El último elemento de nuestro problema es la función $f : \mathcal{X} \to \mathcal{Y}$, que nos dará para un vector de características de un motor una etiqueta según la clase a la que pertenezca.

Tras separar el conjunto de test para evitar el data snooping, se ha tratado de representar gráficamente el conjunto de train con sus clases utilizando la conocida técnica `t-SNE` (t-Distributed Stochastic Neighbor Encoding [5]). Esta técnica requiere una serie de parámetros que dependen de la distribución de los datos y nos ayuda transformar el conjunto multidimensional en un conjunto con las componentes que nos interesen, que para visualización suelen ser 2 ó 3.

Sin embargo, es bien sabido que obtener los parámetros adecuados para que produzca los resultados deseados no es tarea sencilla. De hecho, en [9] se muestra cómo dentro de una misma muestra, según los hiperparámetros que se usen en `t-SNE`, se pueden dar diferentes proyecciones al plano euclídeo.

Aún así, se ha intentado sin éxito probar diferentes combinaciones de parámetros (perplejidad, número de iteraciones,tasa de aprendizaje) para tratar de obtener algo de información sobre los datos, y el resultado es el siguiente:

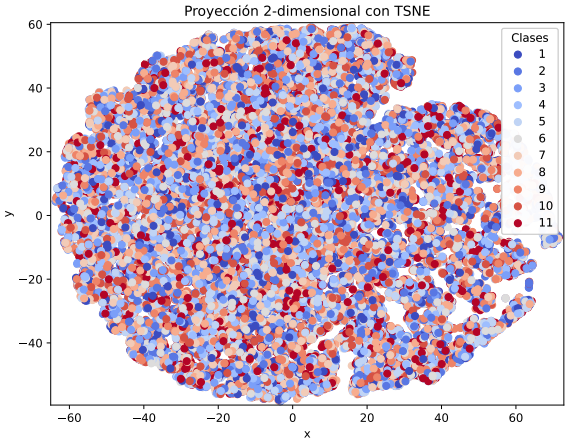

Figura 4: Proyección al plano euclídeo y ajuste de *t-SNE* con parámetros por defecto.

Se han usado los parámetros por defecto para el gráfico pues el resultado es muy similar al que ocurre si se utilizan parámetros más sofisticados.

Lo que sí podemos hacer ahora es un gráfico interesante sobre el número de elementos que tenemos de cada clase. Podemos hacerlo en ambos conjuntos, pues como no estamos mirando los datos como tal sino solo contando el número de etiquetas, no estamos cometiendo *data snooping*.

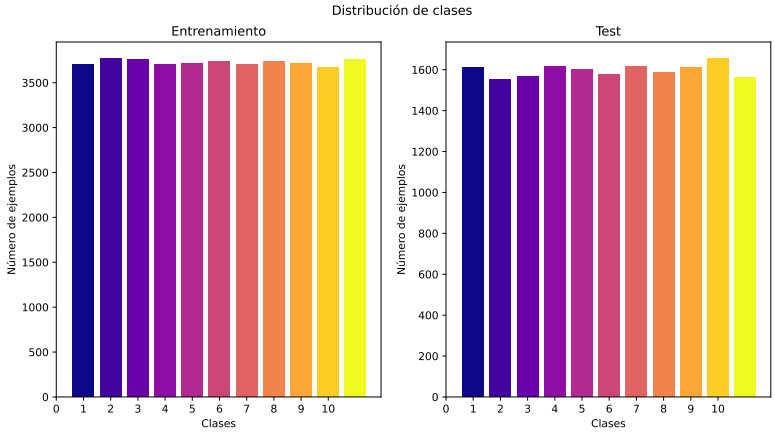

Figura 5: Número de elementos por clase en los conjuntos de entrenamiento y test.

Como podemos ver, el número de elementos que tenemos por cada clase es muy similar en todas las clases. Esto nos indica que no tendremos una muestra desbalanceada y no estará sesgada en el sentido de tener muchos más representantes de unas clases que de otras.

Se ha intentado explorar un poco los datos para ver qué tipo de transformaciones o preprocesado pueden ser buenas para el conjunto de datos. Lo primero que se hace es ver que , como al ver que los datos tienen valores muy pequeños (muy por debajo de 1), se busca si hay columnas de datos que tengan varianza menor a un umbral. Sobre los datos recién llegados, el resultado obtenido es:

```
There are 30 cols with variance lesser than 0.01
```

Este número parece muy elevado , pero tiene sentido debido a los valores tan pequeños que toma nuestro conjunto de datos. Así que pensamos que una parte del preprocesado de datos será de nuevo **estandarizar** nuestros datos.

Se comprueba mediante la creación de un `pipeline` que si se estandarizan los datos y luego se intenta hacer `VarianceThreshold(0.01)` (es decir, eliminar esas columnas que tengan varianza inferior a 0.01), no se elimina ninguna columna, por lo que a priori podríamos decir que todas las variables dan información.

Es conveniente ver también si existen correlaciones entre las variables que tenemos por cada dato, igual que hacíamos en el caso anterior. Esto podría de nuevo darnos pistas sobre si todas las variables que se nos han dado son necesarias para explicar nuestro conjunto. En este caso tenemos menos variables que en el caso anterior, es por ello que podemos visualizar la matriz de correlaciones entre las variables de forma más clara:

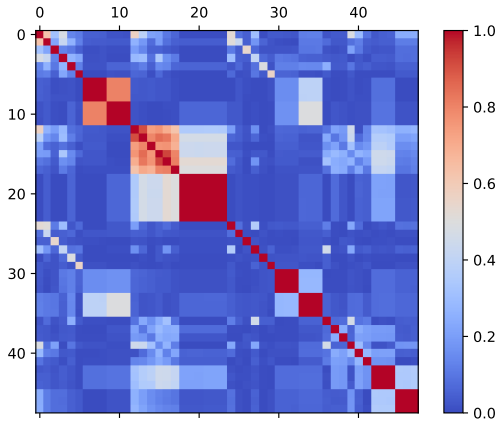

Figura 6: Matriz de correlaciones del conjunto de entrenamiento.

Podemos ver con mucha claridad que hay zonas , por ejemplo entre la variable 18 y la 24 que están completamente correladas. Al igual que esa, existen otros conjuntos de variables que tienen una correlación perfecta, cosa que no nos ocurría con tanta claridad.

Sin embargo, sabemos que correlación no siempre indica causalidad como nos ocurría en el caso anterior, así que tendremos que estudiar si reducir algunas de estas características puede llevarnos a mejores resultados a la hora de entrenar nuestro modelo.

2.2. La clase de funciones $\mathcal{H}$.

Ahora, las funciones deben ser del mismo tipo en el sentido de que en esencia deben ser lineales, es decir,

$$h(x) = w^T x, \quad w \in \mathbb{R}^n$$

Recordamos que cuando tratábamos de clasificar elementos en anteriores prácticas, como solo teníamos dos clases, nombrábamos una como positiva y la otra sería la negativa y podíamos tomar simplemente el signo de $h(x)$ como la etiqueta predicha para un elemento del conjunto.

Ahora, estamos ante un problema de clasificación multietiqueta (en concreto, tenemos 11) etiquetas, por lo que debemos emplear otra estrategia. En este caso , usaremos *one-versus-all* (también conocida como *one-versus-rest*). Debemos hacer un hiperplano w_i para cada clase.

Recordamos siguiendo lo que hemos visto en [1] que, como w es ortogonal a todo vector que esté en el hiperplano, entonces podemos considerar a $h(x)$ una distancia con signo del punto x al hiperplano salvo el cociente por $\|w\|$. En *one versus all*, consideramos que si queremos ver si un elemento $x \in \mathcal{X}$ pertenece a la clase $i-$ésima, se toma esta clase como la clase positiva y todas las demás como negativas, quedándonos así con un problema de clasificación binaria. Hacemos esto para todas las clases y, como

$h_i(x) = w_i^T x$ es una distancia, consideraremos que la clase del elemento x es la que obtenga el valor más grande. En el caso de no haber ninguno, se toma el valor más cercano a la frontera de clasificación. Matemáticamente, podemos expresar esto como:

$$g(x) = \arg\max_i h_i(x).$$

Es por ello que la clase de funciones que obtenemos en nuestro problema es la siguiente

$$\mathcal{H} = \left\{ \arg\max_i w_i^T x \ : \ w_i \in \mathbb{R}^n, i = 0, \dots, 11 \right\}.$$

Igual que en el caso de regresión, no tenemos información suficiente para justificar la realización de transformaciones no lineales del espacio para obtener datos en el $\mathcal{Z}-$espacio que puedan darnos mejores resultados tanto dentro como fuera de la muestra, así que se decide no aplicar esas transformaciones.

2.3. Conjuntos de entrenamiento, validación y test.

En este caso, de nuevo tenemos todos los datos en un único fichero que tenemos que dividir nosotros. Optamos por volver a realizar una partición de 70 % para el conjunto de entrenamiento y 30 % para el conjunto de test.

También volveremos a utilizar en el entrenamiento $K-$*Fold cross validation*, aunque en este caso es más relevante el hecho de que esta división sea estratificada, para mantener la distribución por clases de nuestro conjunto en cada una de las particiones y tener representantes de todas las clases en todos los subconjuntos y que estos no queden sesgados de cara al entrenamiento.

2.4. Preprocesado de datos. Selección de modelos.

El procedimiento que seguiremos en este caso será el de crear diferentes `pipelines` de preprocesamiento de datos con diferentes técnicas de reducción de dimensionalidad para estudiar si alguno de ellos es mejor para este problema, ya que hemos visto en la matriz de correlaciones que puede que haya características que sean suprimibles de cara al entrenamiento.

En uno de de los `pipelines` usaremos PCA, que ya ha sido comentado en la sección de regresión. Este es un algoritmo no supervisado, pues no necesita de las etiquetas para realizar su selección de características, cosa que sí hace el otro selector de características que usaremos:**ANOVA** (*Analysis of variance*. Este selector basa su funcionamiento en el test estadístico *F-test*, que estima el grado de dependencia lineal de las variables dos a dos y posteriormente ordena las variables de la más a la menos discriminativa. Una vez ordenadas, debemos seleccionar un número de variables con las que queremos quedarnos para realizar el entrenamiento.

ANOVA está implementado en `sklearn` en la función `f_classif` del módulo `feature_selection`. Además, usamos luego de este mismo módulo `SelectKBest` para quedarnos con los k primeros para obtener las variables más discriminativas.

Además de esta selección de características previa, realizamos también primero la estandarización que ya hemos realizado anteriormente.

2.4.a. Selección de modelos.

Se han estudiado en la teoría diferentes modelos que podríamos aplicar a este problema. Sabemos que buscaremos modelos lineales que nos darán hiperplanos que separen los datos. Tenemos una selección variada de algoritmos que podríamos utilizar para encontrar el hiperplano que mejor ajuste nuestros

datos, como `PLA-Pocket, LogisticRegression, Hard/Soft SVM ...`. En este caso, vamos a escoger poner a competir los resultados de **Regresión Logística** con los resultados de **SVM**(Support Vector Machine). Recordamos que en ambos tendremo que usar *one-vs-all* para extrapolar la clasificación binaria que nos dan estos métodos al caso multietiqueta.

Regresión logística ya ha sido explicado en cierta profundidad en una de las prácticas anteriores [6] y omitimos por ello su explicación.

Comentamos brevemente el funcionamiento de los `SVM` y por qué los elegimos para el problema. Lo primero es decir que en este caso supondremos que el conjunto de datos no es a priori separable (ya hemos visto que `t-SNE` no consigue separar el conjunto de datos de forma sencilla, aunque podría existir un conjunto de parámetros para el cual sí los separase). Por ello, debemos centrarnos en el caso de *Soft Margin SVM*. Sabemos que los *Hard Margin SVM* trataban de buscar el hiperplano que maximizase el la distancia de los puntos de soporte al hiperplano. En este caso, al ser los datos no separables, tenemos que añadir una penalización ξ, y por tanto nuestro problema de optimización se convierte en resolver el problema de minimización:

$$\begin{cases} & \frac{1}{2}w^T w + C\sum_{n=1}^{N}\xi_n \\ \text{subject to:} & y_n(w^T x_n + b) \geq 1 - \xi_n \ , \ \xi_n \geq 0 \end{cases}$$

En este caso, C es el nivel de penalización que queremos darle a los puntos que atraviesen este margen. Se elige este algoritmo porque al escoger el mejor margen, se trata de hacer que la generalización sea lo mejor posible y además nuestro hiperplano sea más robusto si existe ruido cercano al hiperplano.

2.5. Métricas de error.

En este caso debemos usar una métrica diferente. Nos interesa un error que nos indique, dada una hipótesis h, cuántas veces de media obtenemos clasificaciones erróneas. Claramente, si (x_n, y_n) representa un par: (vector de atributos, etiqueta) , entonces el error que queremos es:

$$E(h) = \frac{1}{N}\sum_{i=1}^{N}[\![h(x_n) \neq y_n]\!].$$

Como $[\![h(x_n) \neq y_n]\!] \in \{0, 1\}$, tenemos que $E(h) \in [0, 1]$ para toda hipótesis h. Este error también tiene ventajas en su interpretación, pues podemos considerar a su vez para una hipótesis $h \in \mathcal{H}$:

$$Acc(h) = 1 - E(h),$$

el acierto medio de la hipótesis.

2.6. Regularización y parámetros del modelo.

2.7. Selección de hipótesis.

2.8. Error final fuera de la muestra.

Referencias

[1] Yaser S. Abu-Mostafa, Malik Magdon-Ismail y Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. ISBN: 1600490069.

[2] 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).

[3] *Hilbert–Huang transform*. en. Page Version ID: 1021767752. Mayo de 2021. URL: https://en.wikipedia.org/w/index.php?title=Hilbert%E2%80%93Huang_transform&oldid=1021767752 (visitado 31-05-2021).

[4] Max Kuhn y Kjell Johnson. *Applied predictive modeling*. 2013. URL: http://www.amazon.com/Applied-Predictive-Modeling-Max-Kuhn/dp/1461468485/.

[5] Laurens van der Maaten y Geoffrey Hinton. "Viualizing data using t-SNE". En: *Journal of Machine Learning Research* 9 (nov. de 2008), págs. 2579-2605.

[6] Javier Sáez. *fjsaezm/ML*. original-date: 2021-03-03T09:34:26Z. Jun. de 2021. URL: https://github.com/fjsaezm/ML/blob/75278b9a95ccb579cfe48c3ef87328296db8faf4/P2/memoria.pdf (visitado 03-06-2021).

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

[8] *UCI Machine Learning Repository: Dataset for Sensorless Drive Diagnosis Data Set*. URL: https://archive.ics.uci.edu/ml/datasets/dataset+for+sensorless+drive+diagnosis (visitado 31-05-2021).

[9] Martin Wattenberg, Fernanda Viégas y Ian Johnson. "How to Use t-SNE Effectively". en. En: *Distill* 1.10 (oct. de 2016), e2. ISSN: 2476-0757. DOI: 10.23915/distill.00002. URL: http://distill.pub/2016/misread-tsne (visitado 02-06-2021).

3. Apéndice

3.1. Resultados de los modelos en Regresión

Se incluyen las tablas con los resultados de los modelos para el problema de regresión. Se incluye una tabla con el preprocesado de solo estandarización y otra con estandarización y PCA.

Regressor	λ	η	max_iter	E_{cv}
SGDRegressor	0.10000	constant	5000	636.93796
	0.10000	constant	10000	636.93796
	0.10000	optimal	5000	2179664857.58336
	0.10000	optimal	10000	473348330.41042
	0.10000	adaptive	5000	364.28276
	0.10000	adaptive	10000	364.28276
	0.01000	constant	5000	612.90878
	0.01000	constant	10000	612.90878
	0.01000	optimal	5000	11859081963.44443
	0.01000	optimal	10000	2425006432.49777
	0.01000	adaptive	5000	329.95897
	0.01000	adaptive	10000	329.95897
	0.00100	constant	5000	651.00770
	0.00100	constant	10000	651.00770
	0.00100	optimal	5000	107610500809.15002
	0.00100	optimal	10000	22183869730.15907
	0.00100	adaptive	5000	314.26437
	0.00100	adaptive	10000	314.26437
	0.00010	constant	5000	662.74988
	0.00010	constant	10000	662.74988
	0.00010	optimal	5000	418399887304.88763
	0.00010	optimal	10000	418399887304.88763
	0.00010	adaptive	5000	312.19840
	0.00010	adaptive	10000	312.19840
Ridge	**0.10000**		**5000**	**310.25456**
	0.10000		10000	310.25456
	0.01000		5000	310.25458
	0.01000		10000	310.25458
	0.00100		5000	310.25665
	0.00100		10000	310.25665
	0.00010		5000	310.25688
	0.00010		10000	310.25688

Tabla 3: Resultados obtenidos según los parámetros usando sólo estandarización.

Regressor	λ	η	max_iter	E_{cv}
SGDRegressor	0.10000	constant	5000	510.56864
	0.10000	constant	10000	510.56864
	0.10000	optimal	5000	473.66396
	0.10000	optimal	10000	473.66396
	0.10000	adaptive	5000	473.59663
	0.10000	adaptive	10000	473.59663
	0.01000	constant	5000	510.15916
	0.01000	constant	10000	510.15916
	0.01000	optimal	5000	468.05363
	0.01000	optimal	10000	468.05363
	0.01000	adaptive	5000	468.02466
	0.01000	adaptive	10000	468.02466
	0.00100	constant	5000	511.68573
	0.00100	constant	10000	511.68573
	0.00100	optimal	5000	472.34160
	0.00100	optimal	10000	472.34160
	0.00100	adaptive	5000	467.97220
	0.00100	adaptive	10000	467.97220
	0.00010	constant	5000	511.73210
	0.00010	constant	10000	511.73210
	0.00010	optimal	5000	498.85089
	0.00010	optimal	10000	498.85089
	0.00010	adaptive	5000	467.97353
	0.00010	adaptive	10000	467.97353
Ridge	0.10000		5000	467.96851
	0.10000		10000	467.96851
	0.01000		5000	467.96852
	0.01000		10000	467.96852
	0.00100		5000	467.96852
	0.00100		10000	467.96852
	0.00010		5000	467.96852
	0.00010		10000	467.96852

Tabla 4: Resultados obtenidos según los parámetros usando estandarización y PCA.