REGRESIÓN: SUPERCONDUCTIVITY DATASET

Descripción del problema

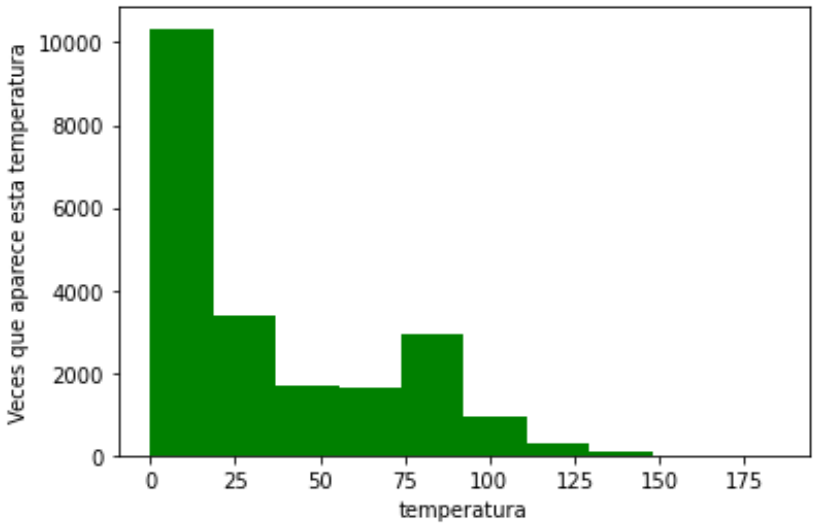
Este problema tiene como objetivo la predicción de la temperatura crítica de algunos materiales. Esta temperatura sería aquella a partir de la cual son superconductores y, para ello, partimos de un conjunto de 21263 datos, uno por cada material que tenemos, con 81 atributos cada uno. Otro dato proporcionado es la fórmula química de cada elemento que, en nuestro caso, no sería necesario y, por tanto, tampoco usado. De esta forma, de los datos a los que tenemos acceso **"./data/regresión/train.csv"** y **"./data/regresión/unique_m.csv"** solo haremos uso de "train.csv".

La organización de los atributos viene dada por 82 columnas, de las cuales, las 81 primeras son información acerca del material y la última se correspondería con la temperatura crítica de este a raíz de sus características previas. Estos valores representan atributos como la masa atómica media, el rango de conductividad térmica, la media de la conductividad térmica, etc. Todos los atributos tienen valores positivos y el dominio de la temperatura crítica no es conocido.

Se trata de un problema de aprendizaje supervisado con variables numéricas del campo de la física. La base de datos es de 2018 y la fuente es Kam Ham idieh (khamidieh@gmail.com) de la Universidad de estadística de Pennsylvania.

Dos buenas formas de abordar los datos serían distinguir aquellos más importantes y analizar la distribución de la temperatura crítica para intentar sacar relaciones o patrones que nos ayuden a ver de qué forma se puede predecir la temperatura de cada material.

En relación con analizar la distribución de los datos, obtenemos que existen muchísimos más materiales con temperatura baja que alta. En la siguiente gráfica veremos en el eje x las temperaturas que se leen del archivo de datos y en el eje y el número de veces que sale la temperatura de ese intervalo. Utilizamos un histograma, que es una representación gráfica que nos permite ver la distribución de las frecuencias de una muestra.



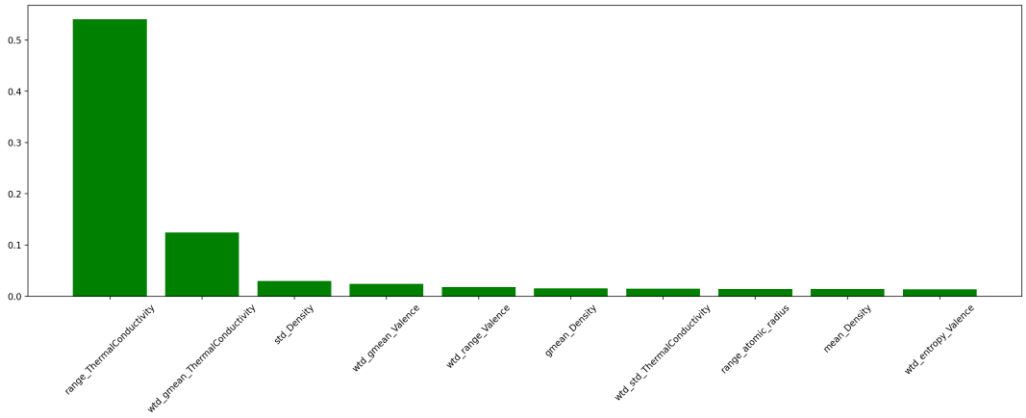
Tras este análisis con el modelo DecisionTreeRegressor() obtenemos que los atributos con mayor importancia son

```
Los atributos más importantes son:

('range_ThermalConductivity', 0.540346354088092)
('wtd_gmean_ThermalConductivity', 0.12466730943940205)
('std_Density', 0.02934888246700974)
('wtd_gmean_Valence', 0.023047284057533778)
('wtd_range_Valence', 0.01770946839460909)
('gmean_Density', 0.015839471285623262)
('wtd_std_ThermalConductivity', 0.015051064732651474)
('range_atomic_radius', 0.014269914643936723)
('mean_Density', 0.013430156039449326)
('wtd_entropy_Valence', 0.013232468636666493)
```

Que aparecen junto a un índice de importancia dentro del conjunto. Como podemos ver, a partir del segundo elemento los valores son realmente parecidos además de bastante inferiores a los dos primeros, por lo que asumiremos que los realmente importantes son *range_ThermalConductivity* **(0.54)** y *wtd_gmean_ThermalConductivity* **(0.1247)**, pero es casi imposible obtener una relación entre estos y el objetivo de predicción.

Vamos a ver una gráfica que muestre la importancia de los 10 datos que más lo son en la que veremos la diferencia entre los dos primeros y el resto.



<u>Selección de hipótesis</u>

Una conclusión que se puede sacar de tratar los datos puede ser que los datos tendrán una distribución no lineal, por lo que usaremos transformaciones cuadráticas para una aproximación correcta de la función de predicción. Al decidir el uso de esta aproximación para obtener mejores resultados debemos tener en cuenta que es casi obligatorio utilizar técnicas de regularización o caeremos en errores de sobreajuste.

Las transformaciones polinómicas se van a realizar al espacio al que se le aplique la reducción de dimensionalidad.

Definición de los conjuntos de training, validación y test

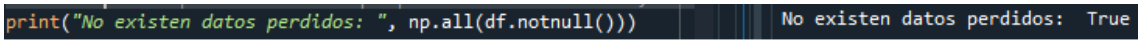
De entrada, los datos vienen en un único conjunto en vez de venir divididos en el conjunto de prueba y de entrenamiento. Por ello, deberemos realizar la división nosotros con la función de **train_test_split**. Se seguirán los parámetros recomendados en clase de un 80% de datos para entrenar y un 20% de datos de prueba de los que no haremos uso hasta el final del proceso. Esto último es debido a que simulamos un problema real en el que no conocemos el conjunto total de datos, sino aquellos con los que entrenamos.

Alguna alternativa podría haber sido mezclar para evitar datos sesgados y escoger los primeros X datos para un conjunto y el resto para el contrario.

El conjunto de validación no tiene que ser definido por el uso de 5-fold CV. En este se harán 5 particiones disjuntas de la parte de los datos que pertenece a Training Data. Cada fold es un bloque de datos y de estos, 4 serán de entrenamiento y el restante de validación. Esta validación es equivalente a hacer test dentro del entrenamiento y todo esto se realiza 5 veces diferentes, por lo que tendremos por cada parámetro al que se lo apliquemos cinco Ein y otros cinco Ecv (equivalente a Eout).

Preprocesado de los datos

Previo a la transformación de cualquier conjunto de datos de los que tenemos realizamos una comprobación de la existencia de valores perdidos. Esto se hace con notnull() aplicado a todos los datos. En la comprobación, y como se ve en la captura de debajo, se ha comprobado que no hay valores de este tipo, por lo que no es necesaria una transformación específica previa, sino que pasamos a preprocesar los datos directamente.



Las fases de preprocesado serán la aplicación al conjunto X de datos de una estandarización, reducimos la dimensionalidad, la transformación polinómica de la que hablamos previamente, eliminar datos según la varianza y una regresión lineal con regularización para evitar el sobreajuste que también será aplicada al conjunto de etiquetas.

El resultado de este preprocesado debería ser que las variables estén bastante menos correladas. Además, tras quitar información redundante, también reducimos el número de atributos inicial que teníamos.

Una definición general del primer paso del preprocesado podría ser que la **estandarización** es el proceso de ajustar o adaptar características en un producto, servicio o procedimiento; con el objetivo de que éstos se asemejen a un tipo, modelo o norma en común. En nuestro caso, escalamos los datos para que estos se ajusten a una distribución normal estándar. Este paso puede no ser necesario para algunos algoritmos, pero para otros es realmente relevante debido a que la varianza de los componentes se ve influida por la escala en la que se encuentren.

El siguiente paso en el que **reducimos la dimensionalidad** se ve justificado por el incremento de precisión que obtenemos al eliminar información con ruido, por la eficiencia que nos aporta al realizar el Training en menos tiempo y por la reducción del sobreajuste tras la eliminación de datos redundantes. Para esta reducción usaremos PCA que permite hallar una cantidad de

factores (menor a los 81 con los que empezamos) que explican de forma cercana el valor de todas ellas para cada uno de los 21263 datos. El uso de esta técnica es uno de los motivos por el que se estandarizan los datos, ya que trabaja con las mayores varianzas de los datos a través de transformaciones lineales. Si al usarlo queremos comprobar que la reducción se ha llevado a cabo, la función shape() nos permite ver la cantidad de elementos que hay en un conjunto por cada dimensión de este. Aplicado al conjunto de datos antes y después del preprocesado vemos que pasamos a tener 35 datos.

El parámetro utilizado para esta función hace referencia al porcentaje de varianza que queremos que PCA utilice y cuanto mayor sea este, menor será la reducción. Como contamos con que los datos tienen mucho ruido, se aplica un valor no muy elevado.

Para las **transformaciones polinómicas** cuadráticas vamos a usar la función PolynomialFeatures que genera una matriz de combinaciones lineales de un grado menor o igual al que le pasemos como parámetro. Como vamos a hacerlo cuadrático, le especificamos que el grado es 2, cuyas entidades son [1, a, b, a^2, ab, b^2].

El último paso sería aplicar **VarianceThreshold** para quitar aquellas variables sin variabilidad. Esto se debe a que, si no la tienen, a nivel estadístico, no existe aprendizaje. Es por ello que variables con valores constantes o con una varianza menor que un umbral especificado deben ser eliminadas.

Medida del error

Como ejemplo se nos permite utilizar MSE, MAE y Accuracy entre otras métricas de error. Para nuestro ejercicio utilizaremos el coeficiente de determinación y MSE.

El **coeficiente de determinación** es un estadístico con el objetivo de probar hipótesis. Este determina qué tan bueno es un modelo replicando resultados y también es conocido como R^2. Los valores obtenidos con este método se encuentran entre 0 y 1.

$$R^2 = \frac{\sum_{t=1}^{T} (\hat{Y}_t - \bar{Y})^2}{\sum_{t=1}^{T} (Y_t - \bar{Y})^2}$$

Tanto el denominador como el numerador corresponden con la expresión de la varianza, pero con la diferencia de que, en este último, la Y con circunflejo es para indicar que es una estimación y que no dividimos entre T porque el denominador también lo estaría y esta se elimina. Este modelo tiene un problema principal que es la no penalización de variables no significativas, pero este problema debería haber sido resuelto con el preprocesado. El modelo es acotado y es el motivo por el que podríamos aplicarlo a diferentes conjuntos.

MSE es también conocido como Error Cuadrático Medio y mide la cantidad de error que existe entre dos conjuntos de datos. Visto de otra forma, mide que tan lejos están los datos obtenidos de los que se esperaba obtener y, cuanto menor sea el valor de este error, más correcto serán los resultados. Como no tiene acotación solamente se aplicará al mismo conjunto de datos.

$$E_{in}(h_w) = \frac{1}{N}\sum_{n=1}^{N}(h_w(x_n) - y_n)^2$$

Parámetros y tipo de regularización usada

Usar regularización consiste en emplear un término que castigue la complejidad del modelo. Según la medida de la complejidad que usemos tendremos diferentes tipos de regularización. En este caso, como hacemos uso de ajustes cuadráticos, hemos aumentado la complejidad del problema y corremos el riesgo de caer en sobreajuste. Minimizando la complejidad minimizamos el coste y obtenemos modelos más simples que generalizan mejor.

El término que se añade con la regularización puede ir multiplicado por un α que indica cómo de importante es que nuestro modelo sea simple, es decir, cuanto mayor sea, más simplificara el modelo. El valor de este componente también puede ser obtenido por cross validation.

Si hacemos uso de la regularización Lasso, medimos la complejidad como la media del valor absoluto de los coeficientes del modelo. La fórmula es:

$$C = \frac{1}{N}\sum_{j=1}^{N}|w_i|$$

Donde C es la complejidad. Esta es aplicable a regresiones tanto lineales como polinómicas, redes neuronales, etc. Este tipo de regularización será útil sobre todo cuando creamos que hay variables de entrada que son poco relevantes.

Por otro lado, la regularización L2 (Ridge) mide la complejidad como la media del cuadrado de los coeficientes del modelo. Puede ser aplicado a varias técnicas como L1 y la fórmula de este quedaría:

$$C = \frac{1}{2N}\sum_{j=1}^{N}w_i^2$$

En este caso, su utilidad brilla cuando tenemos datos de entrada correlados.

Estimación de hiperparámetros y selección de la mejor hipótesis

Para esta parte en la que se prueban modelos existen diferentes alternativas, cada una con sus ventajas y desventajas con respecto de la calidad de los resultados y el tiempo que conllevan.

Para empezar, podríamos hacer el experimento con el conjunto total de datos, lo cual es una idea pésima por el alto optimismo de nuestro error. Por otra parte, podríamos llevar a la práctica el Hold-out separando train y test, entrenando en uno y probando en otro. A pesar de que esta última técnica es mejor que la primera, tiene el problema de ser altamente

dependiente de la partición que se realice, tentándonos a hacer particiones interesadas. Como mejora de esta última tenemos el proceso que vamos a aplicar nosotros en nuestro experimento y que a continuación será detallado en profundidad, k-fold Cross Validation.

Para esto, previamente hacemos una separación del conjunto de datos general en dos partes. Esto ya se ha realizado antes del preprocesado, pero se vuelve a especificar aquí porque es un paso necesario para la aplicación de esta técnica. Cuando tenemos el conjunto de entrenamiento y de prueba debemos saber que este último no debería ser tocado hasta terminar la elección completa y que los datos que usaremos para resolver nuestro problema serán los de entrenamiento. Lo más común para el uso de k-fold c-v es que el valor de la k sea 10 o, como en nuestro caso, 5.

Con k = 5, el proceso consistirá en la realización de 5 pruebas en las que, por cada una, dividiremos el conjunto de entrenamiento en 5 particiones disjuntas de las cuales cuatro serán de entrenamiento y una de validación. Por cada prueba, el conjunto de validación será uno diferente al que lo fue en el resto de casos. Este de validación es el equivalente a hacer test dentro del entrenamiento y, por ello, tendremos para cada caso un Ein y un Ecv (un total de 5 pares de errores).

Cross Validation se puede aplicar a los parámetros que queramos (por ejemplo, Artifical Neuronal Network o SVM) y al terminar, realizamos la media de los k (en este caso 5) Ecv y nos quedamos con el modelo que menor valor tenga.

Existe una opción aún más avanzada que es Leave-one-out, pero cuando tenemos muchos datos es inviable.

Para evitar el data snooping usamos pipelines y GridSearchCV para realizar la elección del mejor modelo a la vez que preprocesamos los datos. El pipeline de preprocesado es utilizado para ver la forma de los datos y realizar el entrenamiento a la misma vez, ya que al final de este se mete un placeholder del sitio en el que queremos que empiece el ajuste de los datos. Es la forma de indicar dónde introducir los diferentes modelos entre los que buscaremos el mejor.

GridSearch, al ser un modelo que puedes entrenar, tiene dentro una función fit que aplicaremos más adelante, pero lo que hacemos previamente es definir el modelo con un pipeline concreto, un espacio de búsqueda, etc.

Uno de los modelos a probar en regresión y siguiendo el orden de explicación seguido en la regularización, será Lasso. Este es un modelo lineal que estima coeficientes dispersos. Es de gran utilidad en situaciones en las que buscamos soluciones con menos coeficientes diferentes de cero, haciendo que la solución dependa de una cantidad de entidades menor. Su función objetivo a minimizar es:

$$\min_{w} \frac{1}{2n_{\text{samples}}} ||Xw - y||_2^2 + \alpha ||w||_1$$

y algunos parámetros que puede recibir son α (explicado en el apartado de regularización), un booleano para normalizar o no, un máximo de iteraciones, entre otros. En nuestro caso le pasamos las constantes de regularización que serán 15 valores diferentes del conjunto [10^{-4}, 10^{4}] y el mismo máximo de iteraciones para los 15 datos.

Otro modelo lineal aplicado será Ridge que minimiza la suma residual penalizada de los cuadrados siguiendo la fórmula:

$$\min_{w} ||Xw - y||_2^2 + \alpha||w||_2^2$$

Los parámetros que recibe son los mismos que Lasso.

Por último, aplicamos el modelo de gradiente descendiente estocástico. Este puede ser utilizado con diferentes tasas de aprendizaje y tipos de regularización, por lo que intentaremos probar todos los parámetros que podamos para encontrar los mejores.

```
SGDRegressor(*, self, loss="squared_loss", penalty="l2", alpha=0.0001, l1_ratio=0.15,
             fit_intercept=True, max_iter=1000, tol=1e-3, shuffle=True, verbose=0,
             epsilon=DEFAULT_EPSILON, random_state=None, learning_rate="invscaling",
             eta0=0.01, power_t=0.25, early_stopping=False, validation_fraction=0.1,
             n_iter_no_change=5, warm_start=False, average=False)
```

De estos posibles parámetros de la función, vamos a definir diferentes valores para *loss (squared_loss y épsilon_insensitive)*, para *penalty (l1 y l2)*, para *alpha (15 valores equidistantes entre [10^{-4}, 10^{4}])* y para la tasa de aprendizaje *(learning_rate = optimal y adaptive).*

Por último, comentar que uno de los argumentos pasados a GridSearchCV será scoring='neg_mean_squared_error' porque este maximiza la métrica y queremos que el error sea el mínimo. Este parámetro es la estrategia para evaluar el rendimiento del modelo validado de forma cruzada en el conjunto de pruebas.

Análisis de resultados

Tras la aplicación de cross validation y la obtención de los errores, hemos comprobado que el mejor de los modelos aplicados ha sido Ridge. Al mostrar los parámetros de los que se han hecho uso vemos que son:

```
--- Pulsar tecla para ver los mejores parametros ---

{'reg': Ridge(alpha=51.79474679231202, max_iter=2500), 'reg__alpha': 51.79474679231202}
```

El máximo de iteraciones ha sido especificado por nosotros y el Alpha obtenida es 51.79, lo cual significa una regularización muy fuerte. Vemos que, durante el proceso de elección del método, el error más bajo que ha alcanzado es 354.97.

```
--- Pulsar tecla para ver el menor error obtenido por el modelo ---

354.97340143295867
```

Lo cual es un valor bastante parecido al obtenido con MSE (que es la métrica que trata de maximizar el GridSearchCV) cuando seleccionamos Ridge como hipótesis final. Como estos errores están al cuadrado, podemos evaluar los errores como la raíz cuadrada de los obtenidos con MSE (18.8 dentro de la muestra y 18.91 fuera de esta)

```
TRAIN
Error de train con MSE:  353.43956425649407
Error de train con R^2:  0.699488350759593
TEST
Error de test con MSE:  357.4069374790853
Error de test con R^2:  0.6924179796534626
```

Por otra parte, comentando el error obtenido con el cociente de determinación, es importante saber que el resultado oscila entre 0 y 1 y, cuanto más cerca de 1 se sitúe su valor, mayor será el ajuste del modelo a la variable que estamos intentando explicar. En nuestro caso, un valor cercano a 0.7 indica una buena calidad del modelo.

CLASIFICACIÓN: SENSORLESS DRIVE DIAGNOSIS DATASET

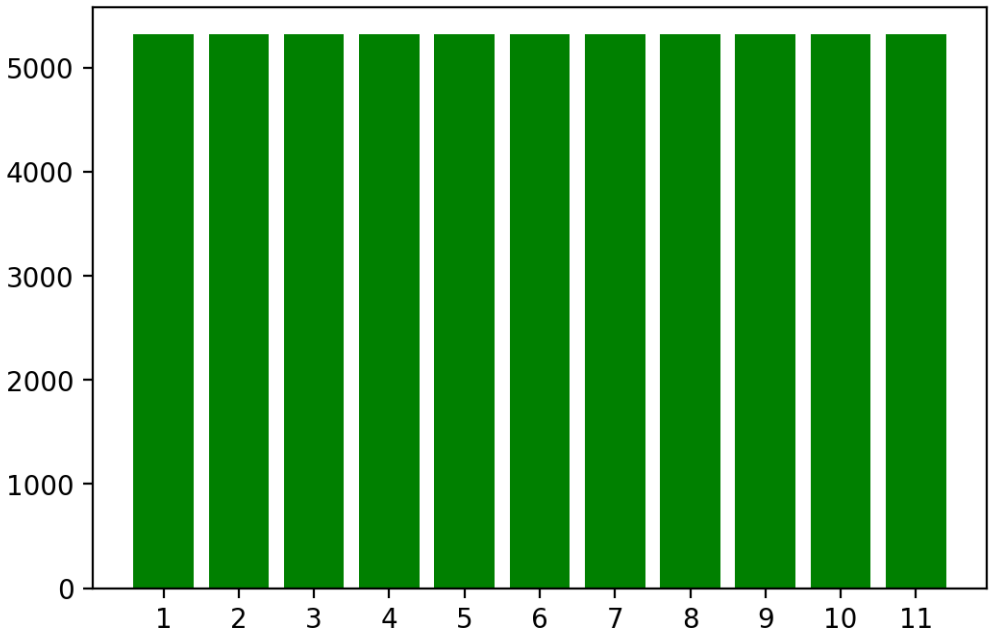
Descripción del problema

Este problema tiene como objetivo clasificar los componentes de un motor que pueden estar intactos o defectuosos. Tenemos datos de 58509 pruebas realizadas en las que cada dato tiene 49 atributos y, de nuevo, el último de ellos hace referencia a la etiqueta correspondiente a cada elemento. Las 11 clases en las que se pueden clasificar los componentes son obtenidas de experimentos con corrientes eléctricas de diferentes condiciones.

Estamos ante un problema de aprendizaje automático no supervisado de clasificación y los datos se nos disponen en un fichero de texto llamado “Sensorless_drive_diagnosis”.

Sobre la base de datos, tanto la titular como donante es Martyna Bator (University of Applied Sciences, Ostwestfalen-Lippe, martyna.bator '@' hs-owl.de).

Para ver los datos con más detalle mostraremos la distribución para comprobar la homogeneidad del conjunto.



Selección de hipótesis

Para el problema de clasificación también aplicaremos transformaciones cuadráticas para obtener un mejor ajuste de los datos. Que el conjunto no sea linealmente separable, a pesar de ser una suposición, nos anima a que apliquemos modelos que se adapten mejor y con la reducción de dimensionalidad conseguimos no aumentar en exceso el tiempo de proceso.

Definición de los conjuntos de training, validación y test

De entrada, estos datos también vienen en un único conjunto en vez de venir divididos en el conjunto de prueba y de entrenamiento. Por ello, deberemos realizar la división nosotros con la función de train_test_split. Se seguirán los parámetros recomendados en clase de un 80% de datos para entrenar y un 20% de datos de prueba de los que no haremos uso hasta el final del proceso. Esto último es debido a que simulamos un problema real en el que no conocemos el conjunto total de datos, sino aquellos con los que entrenamos.

El resto de procedimiento es idéntico al empleado en regresión, por lo que no voy a repetir la explicación. Se aplica cross validation para la obtención del mejor modelo, por tanto el conjunto de validación ya va implícito en su aplicación.

Preprocesado de los datos

El preprocesado es otra parte en la que existen muy pocas diferencias con respecto de regresión.

Las fases de preprocesado volverán a ser la aplicación al conjunto X de datos de una estandarización, reducimos la dimensionalidad, la transformación polinómica de la que hablamos previamente, eliminar datos según la varianza y una regresión lineal con regularización para evitar el sobreajuste que también será aplicada al conjunto de etiquetas.

Esta última se realiza a la vez que la elección del mejor modelo cuando aplicamos GridSearchCV.

El resultado de este preprocesado debería ser que las variables estén bastante menos correladas. Además, tras quitar información redundante, también reducimos el número de atributos inicial que teníamos, pasando de 48 a 27.

Medida del error

La medida del error que aplicamos esta vez es accuracy (también conocida como grado de exactitud). Este método calcula el porcentaje de casos en los que el modelo acierta etiquetando. Su principal problema es que puede aparentar ser correcto cuando en realidad el modelo sea malo.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

La situación en la que más errores genera es cuando las clases están desbalanceadas.

Parámetros y tipo de regularización usada

La regularización de este caso volverá a ser con Lasso y Ridge.

Estimación de hiperparámetros y selección de la mejor hipótesis

El uso de cross validation será de nuevo el que nos de el mejor modelo para este problema de aquellos que probemos. En este caso, los modelos a aplicar son regresión logística, regresión lineal y perceptrón.

Viendo más en profundidad **regresión logística**, esta es una técnica de aprendizaje automático equivalente a una red neuronal con una neurona. Matemáticamente podemos verlo como:

$$y = \sigma(z) = \sigma(WX) = \sigma(\sum(w_i x_i)) = \sigma(\sum(w_0 x_0 + w_1 x_1 + ... + w_n x_n))$$

y podemos distinguir dos partes, una combinación lineal a la izquierda y una aplicación de la función logística a la derecha. La función logística se puede expresar como

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

y sus características son estar acotada entre 0 y 1 y que sus resultados se pueden interpretar como probabilidades. Los parámetros pasados a este método son el castigo de l2 como regularización y un máximo de 2500 iteraciones.

El **perceptrón** es la forma más simple de una red neuronal usada para clasificación con conjuntos linealmente separables. Su forma de actuar es seleccionando aleatoriamente una línea recta para la clasificación y, tras esto, buscar el primer punto mal clasificado. Cuando lo encuentra, modifica la inclinación de la recta para corregir este error. Si el conjunto no es linealmente separable no podría converger, porque a la vez que arregla un punto, estropea otro(s). Es por esto por lo que se limita el número de iteraciones. Para esta técnica los hiperparámetros son los dos tipos de regularización que hemos explicado y los valores de Alpha {0.1, 0.001, 0.0001, 0.00001}.

El último método que vamos a utilizar el **gradiente descendente estocástico** con los = 'hinge' para la clasificación, equivalente al SVM. Para este caso, la única regularización que utilizaremos será l2, la más frecuente para SVM y, para la constante de regularización pasamos 3 valores equidistantes entre [10^{-5}, 10^{5}].

<u>Análisis de resultados</u>

Tras terminar el experimento vemos que el mejor modelo es Regresión Logística con 2500 iteraciones y la regularización l2. En este caso, esta regularización es la única que se había aplicado.

```
--- Pulsar tecla para ver los mejores parametros ---

{'clas': LogisticRegression(max_iter=2500), 'clas__penalty': 'l2'}
```

En el proceso de cross validation, el mejor resultado obtenido con el objetivo de maximizar la métrica de accuracy es 0.604.

```
--- Pulsar tecla para ver el menor error obtenido por el modelo ---

0.6043753028520312
```

El cual es un valor que concuerda bastante con el error obtenido con este modelo dentro y fuera de la muestra de entrenamiento una vez que es seleccionado como hipótesis final.

```
TRAIN
Error dentro de la muestra con accuracy:  0.6001025487640738
TEST
Error fuera de la muestra con accuracy:  0.5931464706887711
```

Este error se puede interpretar como un porcentaje de acierto que, aunque no es necesariamente fiable, un valor cercano al 60% tanto dentro como fuera del conjunto de entrenamiento indica que es buena elección como hipótesis final.