APRENDIZAJE AUTOMÁTICO

Práctica 3

Mario Muñoz Mesa

4 junio 2021

Índice

1. Regresión (Superconductivity)

1.1. Planteamiento.

Suponemos $(\Omega, \mathcal{A}, P)$ espacio probabilístico, donde Ω es el conjunto de todos los posibles superconductores; $\mathcal{A}$ sigma-álgebra formada por todos los subconjuntos de Ω, y P distribución de probabilidad desconocida.

Sobre $(\Omega, \mathcal{A}, P)$ tenemos el vector aleatorio $\mathbf{x} = (x_1, \dots x_{81})$ donde cada variable aleatoria $x_i \colon \Omega \to \mathbb{R}$, $i \in \{1, \dots, 81\}$, mide: el número de elementos para $i = 1$ y características relacionadas con: atomic mass, first ionization energy, atomic radius, density, electron affinity, fussion heat, thermal conductivity o valence, para $2 \leq i \leq 81$. También tenemos la variable aleatoria $y \colon \Omega \to \mathbb{R}$ que asigna la temperatura crítica (grados Kelvin) a cada superconductor (más detalles en el documento adjunto a Superconductivity Data Set). Por lo que $\mathcal{X} = \mathbf{x}(\Omega) = \mathbb{R} \times \overset{81}{\cdots} \times \mathbb{R} = \mathbb{R}^{81}$ y $\mathcal{Y} = y(\Omega) = \mathbb{R}$

A partir del fichero alojado en Superconductivity Data Set se toma $N < 21263$, que será el tamaño de la muestra de entrenamiento. Reservando el 20 % de los datos para test obtuvimos $N = 17010$

En ambos casos, para conseguir estimación g $\mathcal{H}$-lineal de f, y dado que tenemos muestra i.i.d., se seguirá el criterio ERM (minimización de riesgo empírico). Como tenemos que cuidar la cota de error de generalización, aplicaremos regularización para evitar sobreajuste.

1.2. Clase de funciones a usar.

Realizaremos una transformación de segundo orden polinomial a los vectores de características, pues aumenta la flexibilidad de nuestra regresor. Para no aumentar en exceso la complejidad de la clase de funciones (mayor dimensión $\Rightarrow$ mayor complejidad $\Rightarrow$ mayor cota de error de generalización) reduciremos previamente la dimensionalidad de nuestros vectores de características (ver parte de preprocesado). No utilizamos transformación polinómica de mayor orden pues cuanto mayor sea la longitud de los vectores de características mayores posibilidades de disminuir el error en la muestra pero mayores posiblidades de aumentar error en la población (sobreajuste); a parte de incrementar el coste computacional.

Por lo que, como hemos comentado, aplicaremos a $\mathcal{X}$ la transformación Φ_2, que genera combinaciones polinómicas de grado menor o igual que 2 de las características

$$\Phi_2(\mathbf{x}) = (1, x_1, \dots, x_{\hat{d}}, \underbrace{x_1x_2, \dots, x_1x_{\hat{d}}, x_2x_3, \dots, x_2x_{\hat{d}}, \dots x_{\hat{d}-1}x_{\hat{d}}}_{\text{combinaciones } x_ix_j \text{ con } i<j, \quad i,j\in\{1,\dots,\hat{d}\}}, x_1^2, \dots x_{\hat{d}}^2)^T$$

Nota: aquí $\hat{d} < d = 81$ pues no utilizaremos todas las características. Podemos ver que $\Phi_2(\mathbf{x})$ tiene $1 + \hat{d} + (\sum_{i=1}^{\hat{d}-1} \hat{d} - i) + \hat{d} = 1 + 2\hat{d} + (\hat{d} - 1)\hat{d} - \frac{(\hat{d}-1)\hat{d}}{2} = 1 + 2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2}$ componentes

Denotando con d a $2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2}$, la clase de funciones hipótesis a usar será

$$\mathcal{H} := \{h_w \colon \mathbb{R}^{d+1} \to \mathbb{R} \ : \ h_w(\Phi_2(\mathbf{x})) = w^T\Phi_2(\mathbf{x}), \ w \in \mathbb{R}^{d+1}\}$$

Denotando con $\mathbf{x}_i$ a $\Phi_2(\mathbf{x}_i)$, $i \in \{1, \dots, N\}$, el error en la muestra a minimizar es

$$E_{in}(w) := \frac{1}{N}\sum_{i=1}^{N}(w^T\mathbf{x}_i - y_i)^2$$

es decir, la media del error cuadrático en cada uno de los elementos de la muestra. Equivalentemente, usando notación matricial

$$E_{in}(w) = \frac{1}{N}\|Xw - y\|^2$$

donde

$$X = \begin{pmatrix} -\mathbf{x}_1^T- \\ -\mathbf{x}_2^T- \\ \dots \\ -\mathbf{x}_N^T- \end{pmatrix} \qquad y = \begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_N \end{pmatrix}$$

Queremos minimizar

$$E_{in}(w) = \frac{1}{N}\|Xw - y\|^2 = \frac{1}{N}(Xw-y)^T(Xw-y) = \frac{1}{N}((Xw)^TXw - y^TXw - (Xw)^Ty + y^Ty1) =$$

$$= \frac{1}{N}(w^TX^TXw - 2w^TX^Ty + y^Ty)$$

es decir, buscamos vector de pesos $\hat{w} \in \mathbb{R}^{d+1}$ con $E_{in}(\hat{w}) = \text{mín}_{w\in\mathbb{R}^{d+1}} E_{in}(w)$. El gradiente queda

$$\nabla E_{in}(w) = \frac{2}{N}X^T(Xw-y) = 2X^TXw - 2X^Ty$$

Técnicas conocidas para obtener mínimo de E_{in} en regresión lineal:

Gradiente Descendente:

Gradiente Descendiente es una técnica general para obtener mínimos de una función derivable. Una analogía ilustrativa es una bola rodando por una superficie con colinas. Si la bola se emplaza en la colina, esta desciende hasta encontrar un valle (mínimo local). Igual ocurre con $E_{in}(w)$, que se puede representar como una superficie de altas dimensiones.

En el comienzo del algoritmo empezamos en un punto w_0 del dominio de E_{in}.

Tenemos que determinar cómo bajar por E_{in} mediante "el paso más profundo". Supongamos que tomamos un paso de tamaño η (tasa de aprendizaje) en la dirección de un vector unitario $\hat{v}$, $\|\hat{v}\| = 1$. El nuevo valor del dominio de E_{in} a utilizar será $w_0 + \eta\hat{v}$. Queremos, dado w_0, encontrar $\hat{v}$ tal que

$$E_{in}(w_0 + \eta\hat{v}) < E_{in}(w_0)$$

Se toma η pequeño, $\eta \approx 0$. Aplicaremos desarrollo de Taylor en la siguiente expresión

$$\Delta E_{in} = E_{in}(w_0+\eta\hat{v}) - E_{in}(w_0) \overset{Taylor}{=} \eta\nabla E_{in}(w_0)^T\hat{v} + \underbrace{\mathcal{O}(\eta^2)}_{\text{O-grande}} \geq \eta\nabla E_{in}(w_0)^T\hat{v} \overset{(*)}{=}$$

$$\overset{(*)}{=} -\eta\|\nabla E_{in}(w_0)\|$$

(*) En la última igualdad tenemos el producto del vector fila $\eta\nabla E_{in}(w_0)^T$ (que es evaluar w_0 en el gradiente de E_{in} como fila), y por otro lado el vector que yo quisiera elegir $\hat{v}$. Para que este producto escalar sea máximo $\hat{v}$ debe tener misma dirección y sentido que $\eta\nabla E_{in}(w_0)^T$, así el coseno del ángulo que forman será 1. Pero como lo que nosotros buscamos es minimizar, tomaremos sentido opuesto para que el producto sea negativo (notar de nuevo que buscamos $E_{in}(w_0+\eta\hat{v}) < E_{in}(w_0)$). Entonces $\eta\nabla E_{in}(w_0)^T$ y $\hat{v}$ coinciden en la misma dirección pero con sentidos opuestos, es decir, $\hat{v}$ tiene la dirección y sentido de $-\nabla E_{in}(w_0)$. Finalmente, basta hacer el producto escalar de $\eta\nabla E_{in}(w_0)^T$ y $\hat{v}$ para ver la igualdad

$$\eta\nabla E_{in}(w_0)^T\hat{v} \overset{(*)}{=} \eta\|\nabla E_{in}(w_0)^T\|\|\hat{v}\|\cos(\pi) = -\eta\|\nabla E_{in}(w_0)\|$$

$$\Leftrightarrow \nabla E_{in}(w_0)^T\hat{v} = -\|\nabla E_{in}(w_0)\| \Leftrightarrow \frac{\nabla E_{in}(w_0)^T\hat{v}}{\|\nabla E_{in}(w_0)\|} = -1 \Leftrightarrow \hat{v} = -\frac{\nabla E_{in}(w_0)}{\|\nabla E_{in}(w_0)\|}$$

Por tanto

$$\hat{v} = -\frac{\nabla E_{in}(w_0)}{\|\nabla E_{in}(w_0)\|}$$

Una vez que está definido el vector que usaremos para desplazarnos en busca de un mínimo, podemos describir el proceder del algoritmo. Se toma una tasa de aprendizaje η, un punto inicial $w = w_0 \in \text{Dominio}(E_{in})$, y se actualiza el valor de w repetidamente

$$w := w - \eta \nabla E_{in}(w)$$

repetimos esta actualización hasta una condición de parada, ya sea por iteraciones o por encontrar un valor suficientemente pequeño.

Gradiente Descendente Estocástico:

Este método, a diferencia de *Gradiente Descendente*, trabaja con minibatches (pequeñas submuestras de la muestra). Está comprobado empíricamente que el uso de estos minibatches para el cálculo del gradiente mejora el óptimo obtenido para funciones no convexas. Al trabajar con submuestra pequeña es probable que en los pasos o desplazamientos del algoritmo se evite caer en mínimos locales no deseables (valor alto), mínimos locales en los que sí se entraría si trabajasemos con toda la muestra.

Se toma una tasa de aprendizaje η, un punto inicial $w = w_0 \in \text{Dominio}(E_{in})$. Se divide la muestra en una secuencia de minibatches aleatorios. Iteramos los minibatches y en base a cada minibatch, $minibatch \in Minibatches$, se actualiza el valor de w

$$w := w - \eta \nabla E_{in}^{minibatch}(w)$$

repetimos esta división en minibatches y los cálculos de w por cada $minibatch \in Minibatches$ hasta una condición de parada, ya sea por iteraciones o por encontrar un valor suficientemente pequeño para E_{in}

Pseudoinversa (solución analítica):

Mediante este método obtenemos analíticamente la solución óptima. Para minimizar E_{in} igualamos gradiente a 0

$$\nabla E_{in}(w) = \frac{2}{N} X^T (Xw - y) = 2X^T Xw - 2X^T y = 0$$

y queda

$$X^T Xw = X^T y$$

que se reescribe como

$$w = X^\dagger y \quad \text{donde} \quad X^\dagger := (X^T X)^{-1} X^T \quad \text{(pseudo-inversa de } X\text{)}$$

$\Rightarrow w_{lin} = X^\dagger y$. Por tanto, dados X e y, el algoritmo consiste en calcular $X^\dagger = (X^T X)^{-1} X^T$ y devolver $w_{lin} = X^\dagger y$

Para calcular $(X^T X)^{-1}$ se puede usar la descomposición en valores singulares (SVD), $X = UDV^T$, y queda $(X^T X)^{-1} = VD^* V^T$ (si D tiene $e_0, \ldots, e_d$ como elementos de la diagonal, la matriz D^* tiene, en la diagonal, $\frac{1}{e_i^2}$ como elementos o 0 si $e_i = 0$)

Nota: Una desventaja de este método es que el cálculo matricial hace que no sea un método con buena escalabilidad, grandes cantidades de datos pueden dar lugar a matrices no abordables.

1.3. Hipótesis finales que se usarán.

En el preprocesado: (1) se eliminan características con coeficiente de correlación lineal en valor absoluto mayor a 0.95, (2) se eliminan las características con varianza 0, (3) se seleccionan las combinaciones de características mediante PCA que expliquen al menos el 97 % de la varianza (4) se realiza transformación polinomial de segundo orden a cada vector de características, (5) se normalizan las características (media 0 y varianza 1).

La métrica a valorar es ECM. Evaluaremos desempeño de los métodos `SGDRegressor` (Gradiente Descendente Estocástico), `Ridge` (sol analítica SVD) (regularización l_2 en ambos) y `Lasso` (descenso de coordenadas) (regularización l_1), para cada uno evaluaremos los parámetros de regularización 0.00001, 0.001 y 0.1 mediante 5-fold cross validation

1.4. Generación de conjuntos training y test.

Hay un compromiso en la selección del tamaño de entrenamiento y test: el tamaño de la muestra de entrenamiento determina el número de instancias que tendremos para entrenar el modelo y por tanto para minimizar E_{in}, y el tamaño del conjunto de test condicionará la estimación, E_{test}, de E_{out} que obtengamos. Evitaremos proporciones extremas de tamaño para test. Suele ser habitual reservar un 20 % de las instancias para test, y eso hemos hecho. De las 21263 instancias tendremos 17010 para test, el resto para training y validación cruzada.

Utilizaremos k-fold cross validation con $k = 5$ para la selección de modelos. Esta técnica es derivada de *Leave-one-out*, la cual nos proporciona una estrategia para abordar la problemática o compromiso de elección del tamaño de conjunto de validación, K, esta es: $E_{out}(g) \approx E_{out}(g^-)$ cuando K pequeño y $E_{out}(g^-) \approx E_{val}(g^-)$ cuando K grande.

La técnica k-fold cross validation consiste en dividir la muestra de entrenamiento en k particiones, cada una de tamaño aproximado $\frac{N}{k}$, e ir iterando sobre ellas eligiendo cada vez una partición para actuar como conjunto de validación y entrenando sobre el resto de la muestra el modelo; una vez iteradas sobre todas las particiones se toma la media de errores obtenidos y ese es el error de validación cruzada E_{cv} que es un buen estimador de E_{out} cuando k es grande, cuanto menor sea k peor será la estimación de E_{out}. Se elige el modelo con menor E_{cv}

Se podría tomar $k = N$ (Leave-one-Out) pero esto implica un alto coste computacional; es por esto que se elige $k << N$, en nuestro caso $k = 5$ (es habitual $5 \leq k \leq 10$)

1.5. Detalles del preprocesado de datos.

Pasos realizados:

1. Se eliminan las características que tienen valor absoluto de coeficiente de correlación lineal mayor a 0.95 con otra característica.

   Esto se hace porque se observó la matriz de coeficientes de correlación en valores absolutos

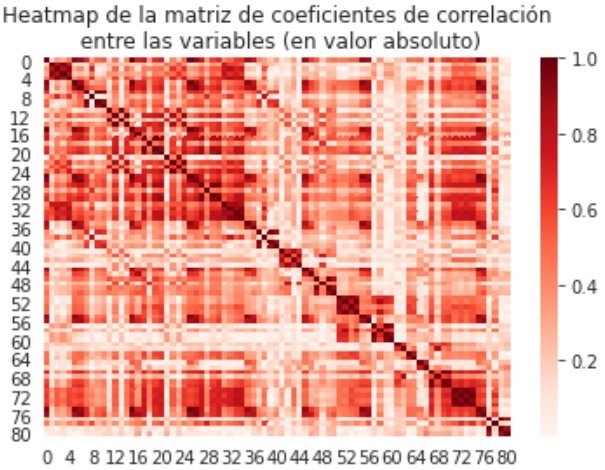

Figura 1: Matriz de coeficientes de correlación en valor absoluto

Las características $4, 9, 13, 14, 15, 19, 24, 25, 26, 29, 30, 34, 39, 49, 54, 59, 69, 70, 73, 74, 75, 76, 79$ tenían valores de coeficiente de correlación lineal mayor a 0.95 en la matriz. Se puede decir que estas características no nos aportan información nueva, además aumentan la dimensionalidad y por tanto empeoran la cota de error de generalización. Se decide eliminarlas y reducir así la complejidad de la clase de funciones. Tras eliminarlas se obtiene

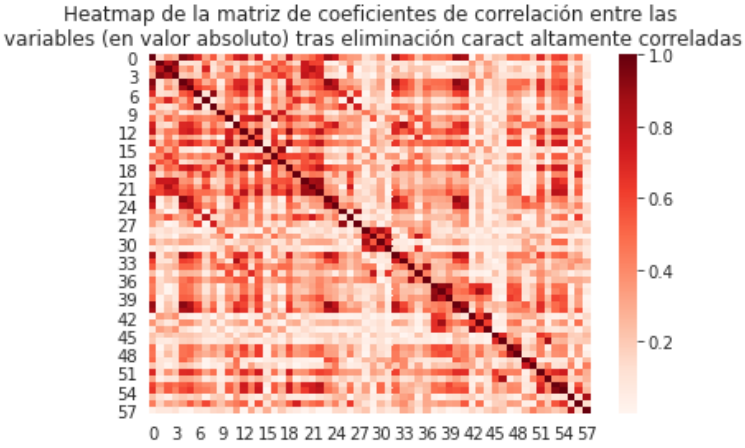

Figura 2: Matriz de coeficientes de correlación en valor absoluto tras la eliminación de características altamente correladas linealmente

una matriz de menor dimensión y con menor apreciación de zonas correspondientes a alta correlación (excepto en diagonal donde tenemos coeficientes de correlación 1 pues es la correlación de cada característica consigo misma).

2. Eliminamos las características con varianza 0 si las hubiese, no aportan ninguna información y además es necesario eliminarlas para después normalizar.

3. Normalizamos las características (media 0 y varianza 1) para el correcto funcionamiento de PCA

4. Seleccionamos las combinaciones de características mediante PCA que expliquen al menos el 97 % de la varianza. Mediante este método esperamos reducir la dimensionalidad con la mayor información posible; de modo que cuando apliquemos la transformación polinómica no tengamos una dimensionalidad excesiva (dimensionalidad muy alta aumenta coste computacional

y mayor cota de error de generalización). La elección de este método ha sido arbitraria, se podría haber elegido cualquier otro método para reducir reducir la dimensionalidad y complejidad de la clase de funciones.

5. Realizamos transformación polinomial de segundo orden a cada característica para aumentar así la flexibilidad de nuestro regresor.

6. Normalizamos las características (tendrán media 0 y varianza 1), no normalizar puede degradar el desempeño de métodos como Lasso y SGD por ejemplo. También deben estar normalizadas de cara a que nuestra regularización actúe correctamente. Normalizamos para evitar estas problemáticas y que todas las características "sean valoradas en la misma magnitud".

1.6. Métrica de error a usar.

La métrica que utilizaremos es el error cuadrático medio, ECM, $ECM = \frac{1}{N}\sum_{i=1}^{N}(h_w(x_i)-y_i)^2$ que nos da la media de la suma de los errores cuadráticos de las diferencias entre las predicciones y las verdaderas etiquetas. Tiene como desventaja que no está acotada superiormente y no tenemos por tanto una referencia de cuánto de malo es un resultado en concreto. Aún así es de uso extendido, es la misma función que se minimiza, y nos servirá para comparar los modelos.

Otra métrica conocida consiste en tomar valor absoluto en la diferencia entre predicciones y verdaderas etiquetas en vez de elevar al cuadrado la diferencia. En este caso tendríamos la media de los valores absolutos de las diferencias (Error Medio Absoluto), sin embargo no consideramos que sea de interés pues usándola daríamos menos importancia a los errores altos que con ECM.

1.7. Parámetros usados y tipo de regularización elegida.

La regularización nos ayudará a evitar sobreajuste, sobretodo después de la transformación polinómica que hemos realizado. Con la regularización se aumentará ligeramente el sesgo para decrementar significativamente la varianza.

Tenemos cierta confianza en haber eliminado características redundantes en el preprocesado. Una hipótesis será asumir que esto ha ocurrido y que la mayoría de nuestras características son informativas y tienen un impacto en el etiquetado. Bajo esta hipótesis utilizaremos dos modelos con regularización Ridge, por lo que tendremos en ellos error aumentado

$$E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2 = E_{in}(\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$$

donde $\lambda \geq 0$ es el parámetro de regularización (este tipo de regularización penaliza los coeficientes de $\mathbf{w}$ grandes).

Por otra parte, estamos trabajando en un problema con bastantes características y de diversa índole (varias caract. sobre radio atómico, varias sobre temperatura de fusión...), y mediante PCA reducimos dimensionalidad teniendo solo en cuenta las características y no el etiquetado. La otra hipótesis será que aún queda un número significativo de características redundantes. Bajo esta hipótesis utilizaremos un modelo con regularización Lasso (regularización Lasso suele funcionar bien cuando hay bastantes caract. poco informativas y regularización Ridge cuando son pocas las características poco informativas), por lo que tendremos error aumentado

$$E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \lambda\|\mathbf{w}\|_1$$

donde $\lambda \geq 0$ es el parámetro de regularización. Con regularización Lasso penalizaremos las características poco informativas (cuando se minimiza con este tipo de regularización tiende a poner coeficientes de $\mathbf{w}$ a 0 en las características poco informativas), si hubiese un número significativo de caract. redundantes esperamos obtener mejor resultado que con regularización Ridge.

Para técnica Gradiente Descendente Estocástico se utiliza la función `SGDRegressor`. Ésta utiliza tamaño minibatch 1. Es interesante notar que la función de error junto con el término de regularización Ridge es convexa; pierde relevancia el uso de minibatches o punto de inicio al no tener óptimos locales. Se podría utilizar Gradiente Descendente sin ningún problema. Aún así, dado que `sklearn` nos proporciona método para la versión estocástica, y que realmente ganamos eficiencia computacional al calcular gradiente en un solo punto, y el ruido que supone computar el gradiente en un solo punto termina promediándose en número alto de iteraciones; se decide utilizar la versión que nos facilita `sklearn`. El learning rate se ha elegido adaptativo, cuando no se esté mejorando en el criterio de tolerancia tras `n_iter_no_change` épocas, entonces actualizamos la tasa de aprendizaje mediante $learning\_rate = \frac{learning\_rate}{5}$, vamos disminuyendo conforme nos acercamos al mínimo para evitar oscilación. Elegimos `n_iter_no_change`=1 y una tolerancia `tol` muy baja; si tras una época no hemos conseguido reducir una cantidad ínfima el error, entonces estaremos oscilando por learning rate muy alto, y por tanto se decrementará. El algoritmo parará cuando si llegamos a learning rate 1e-6 o por iteraciones máximas. Se adjunta código con comentarios del resto de parámetros:

```
{"model": [SGDRegressor(loss = 'squared_loss', # función de pérdida cuadrática
                        penalty = 'l2', # utilizaremos regularización l2
                        # alpha  constante del término de regularización (probaremos distintos valores mediante 5-fold cross validation)
                        fit_intercept = True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                        max_iter = 4100, # Número máximo de iteraciones arbitrario
                        tol = 0.000001, # Tolerancia para criterio de parada por tolerancia (parar si loss > best_loss - tol tras n_iter_no_change épocas seguidas)
                                        # el criterio valora si el error es no mejora en 0.001 el mejor error hasta el momento. En nuestro caso esta condición por tolerancia se usará para learning_rate adaptativo
                        shuffle = False, # no nos interesa introducir más ruido
                        random_state = 1, # para tener reproducibilidad de los resultados
                        learning_rate = 'adaptive', # si no se mejora resultado por criterio tolerancia (loss > best_loss - tol) tras n_iter_no_change épocas seguidas, entonces cambiamos learning rate por (learning rate)/5
                                                    # queremos evitar oscilación
                        eta0 = 0.05, # learning rate inicial arbitrario, nos permitimos que sea un poco alto pues es adaptativo
                        early_stopping = False, # False pues no queremos reservar más datos para validación
                        n_iter_no_change = 1, # cada época sin mejora en crit. tolerancia se realizará la adaptación del learning rate (learning_rate='adaptive')
                        average = False, # no nos interesa obtener media de pesos
                        verbose = 0, # no nos interesan mensajes
                        warm_start = False, # no reutilizamos ninguna solución anterior durante la validación cruzada
                        l1_ratio = 0 # 0 corresponde a l2 penalty, no se usará pues solo se usa si learning_rate = 'elasticnet'
                        )],
      "model__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados


```

Listing 1: Parámetros usados en `SGDRegressor`

Otro de nuestros modelos consiste en obtener una solución analítica mediante descomposición en valores singulares, para ello se utiliza la función `Ridge` indicando método de resolución "svd" que nos dará la solución solución óptima de la función de error con regularización Ridge (error aumentado) haciendo uso de la descomposición en valores singulares de X. Se adjunta código con comentarios sobre los parámetros elegidos:

```
{"model": [Ridge(# alpha  constante del término de regularización (probaremos distintos valores mediante 5-fold cross validation)
                        fit_intercept = True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                        normalize = False, # ya hemos normalizado en preprocesado
                        copy_X = True, # no nos interesa sobreescribir X
                        max_iter = None, # pues resolveremos mediante método analítico
                        # tol lo podemos dejar por defecto pues no se va a usar
                        solver = 'svd', # resolveremos de forma analítica usando descomposición en valores singulares
                        random_state=None)], # no nos hace falta para tener reproducibilidad de los resultados pues no se va a usar solver=sag o solver=saga. Obtendremos solución analítica
          "model__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados


```

Listing 2: Parámetros usados en `Ridge`

Nota: sabemos que, permitiéndolo el tamaño del dataset, mediante un método analítico siempre obtendremos la solución óptima mientras que por Gradiente Descendente Estocástico obtendremos una estimación del mínimo. Aún así, para el propósito de la práctica, mantenemos el de Gradiente Descendente Estocástico, utilizaremos ambos y así también comprobaremos lo comentado.

Por último, para nuestro modelo con regularización Lasso, utilizamos la función `Lasso`, que utiliza algoritmo de descenso de coordenadas en lugar de Gradiente Descendente Estocástico (va actualizando un parámetro cada vez en lugar de todos como en Gradiente Descendente Estocástico). Adjuntamos código con los parámetros comentados:

```
{"model": [Lasso(# alpha  constante del término de regularización (probaremos distintos valores mediante 5-fold cross validation)
                        fit_intercept = True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                        normalize = False, # ya hemos normalizado en preprocesado
                        precompute = False, # tampoco tenemos demasiados datos
                        copy_X = True, # no nos interesa sobreescribir X
                        max_iter = 3000, # número máximo de iteraciones arbitrario
                        tol = 0.0001, # el por defecto, parece una tolerancia razonable. Elección arbitraria
                        warm_start = False, # no reutilizamos soluciones anteriores como inicio
                        positive = False, # no tenemos necesidad de que los coef. sean positivos
                        random_state = 1, # para tener reproducibilidad de los resultados
                        selection = 'cyclic')], # no tenemos tol > 1e^-4 para que pueda resultar interesante 'random'
          "model__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados


```

Listing 3: Parámetros usados en `Lasso`

Nota en general: el número de iteraciones, `n_iter_no_change` y tolerancia se han elegido de forma arbitraria y comprobando que hubiese convergencia. Estos parámetros se podrían haber añadido como parámetros a valorar y expandir nuestro número de modelos para validación cruzada, pero los tiempos de ejecución serían inabarcables.

1.8. Selección de la mejor hipótesis y E_{out}

Para la selección de la mejor hipótesis o modelo utilizaremos 5-fold cross validation, técnica ya explicada en la sección de *Generación de conjuntos training y test*. Elegiremos como modelo ganador aquel con menor E_{cv}. El modelo ganador se entrenará finalmente sobre toda la muestra.

Para hacer esto se ha utilizado `GridSearchCV`, función que nos permite entrenar el modelo ganador en toda muestra mediante `refit=True`, y nos facilita variables con los resultados obtenidos. La métrica considerada para elección del mejor modelo es ECM (error cuadrático medio)

Vamos a valorar distintos parámetros de regularización: tanto para `SGDRegressor`, `Ridge` y `Lasso` tendremos los parámetros de regularización 0.00001, 0.001 y 0.1, por lo que en total tendremos 9 modelos para 5-fold cross validation (se podrían tener más modelos considerando otros parámetros, nos hemos limitado a los parámetros de regularización indicados)

Nos referiremos a los modelos o hipótesis finalmente determinados por `SGDRegressor`, `Ridge` y `Lasso` (junto sus parámetros) mediante $M_{SGDRegressor}$, M_{Ridge} y M_{Lasso} (representan todos los pasos realizados: preprocesado, clase de funciones elegida, parámetros...). Tendremos 9 modelos determinados finalmente por los parámetros de regularización:

$$\{(M_{SGDRegressor}, \lambda), (M_{Ridge}, \lambda), (M_{Lasso}, \lambda)\} \quad \text{con} \quad \lambda \in \{0.00001, 0.001, 0.1\}$$

Veamos los ECM medios de cada modelo obtenidos por 5-fold cross validation

Modelo	ECM medio 5-fold cross validation (E_{cv})
$(M_{SGDRegressor}, 0.00001)$	266.01612282
$(M_{SGDRegressor}, 0.001)$	262.30992773
$(M_{SGDRegressor}, 0.1)$	297.55736393
$(M_{Ridge}, 0.00001)$	261.96442557
$(M_{Ridge}, 0.001)$	261.96439522
$(M_{Ridge}, 0.1)$	261.96137422
$(M_{Lasso}, 0.00001)$	261.96321889
$(M_{Lasso}, 0.001)$	261.85022276
$(M_{Lasso}, 0.1)$	270.32063746

Cuadro 1: ECM medios por 5-fold cross validation de cada modelo

Nuestro mejor modelo con ECM por 5-fold cross validation 261.85022276 es $(M_{Lasso}, 0.001)$. En $M_{SGDRegressor}$ vemos buen resultado en ECM con un valor intermedio de regularización, poca regularización no da buen resultado. Y al aumentar valores de λ estamos imponiendo restricción cada vez más fuerte y el modelo deja de ajustar bien como podemos observar, para $(M_{SGDRegressor}, \lambda)$ con $\lambda = 0.1$ obtenemos ECM 297.55736393, bastante peor.

En M_{Ridge} podemos observar las soluciones óptimas para regularización Ridge, es destacable que el ECM no aumenta conforme incrementamos el valor de regularización, es más, el mejor resultado lo obtenemos con el mayor parámetro de regularización, $\lambda = 0.1$. Esto puede sugerir que en `SGDRegressor`, al modificar la función de error a minimizar (por aumentar λ) los parámetros dejan de ser adecuados y no llegamos a las soluciones óptimas.

Finalmente, en M_{Lasso} vemos que obtenemos la mejor solución con el valor intermedio de los parámetros de regularización, $\lambda = 0.001$. Con Lasso obtenemos una buena solución, de hecho la mejor en nuestra validación cruzada, lo cual podría sugerir que sí había características redundantes, aunque es difícil concluir nada pues nuestro grid es algo limitado y la diferencia de ECM con los modelos M_{Ridge} es bastante baja.

Entrenamos nuestro modelo ganador $(M_{Lasso}, 0.001)$ con todos los datos de entrenamiento. Al utilizar todos los datos de entrenamiento tenemos la ventaja de tener un tamaño de entrenamiento mayor (en 5-fold cross validation no utilizábamos N instancias, siempre reservábamos una de las 5 particiones para validar). Entrenando sobre toda la muestra de entrenamiento se espera ahora obtener un "mejor regresor".

Utilizaremos nuestro conjunto de test que guardamos desde el principio para estimar el error de generalización, estimaremos E_{out} mediante E_{test}. Procediendo así, estimamos, de nuestro mejor modelo $(M_{Lasso}, 0.001)$, ECM 261.58582397 fuera de la muestra; estimación ligeramente más optimista que la que obteníamos en validación cruzada (261.85022276), lo cual va en consonancia con el resultado teórico $E_{out}(g) \leq E_{cv}$ (va en consonancia en el sentido de que $E_{out}(g) \approx E_{test}(g) \leq E_{cv}$), que nos viene a decir que el error de validación cruzada, E_{cv}, nos da una estimación pesimista del error de generalización.

1.9. E_{out} para la mejor hipótesis usando todos los datos para entrenar.

Ahora utilizamos todos los datos disponibles para entrenar nuestro mejor modelo, esto nos dará un mejor ajuste. Para estimar E_{out} ya no disponemos de conjunto de test. Lo que se hará es hacer uso de la cota pesimista de que nos proporciona E_{cv}, tendremos una cota superior de $E_{out}(g)$. La cota será más ajustada cuanto mayor sea k en k-fold cross validation, idealmente $k = N$ (Leave-one-out); de nuevo, por el alto coste computacional que esto supondría, nos conformaremos con $k = 20$

Procediendo así, estimamos $E_{out}(g) \leq 259.9156633155395$. Al estar usando todos los datos disponibles para entrenar estamos obteniendo un "mejor regresor", obtenemos como cota pesimista de E_{out} un valor menor que la estimación que teníamos de E_{out} por E_{test} cuando solo usábamos el conjunto de training para entrenar (que a su vez era menor que el error de validación cruzada).

2. Clasificación (Sensorless Drive Diagnosis)

2.1. Planteamiento.

Suponemos $(\Omega, \mathcal{A}, P)$ espacio probabilístico, donde Ω es el conjunto de posibles configuraciones, de componentes intactas y defectuosas, de motor; $\mathcal{A}$ sigma-álgebra formada por todos los subconjuntos de Ω, y P distribución de probabilidad desconocida.

Sobre $(\Omega, \mathcal{A}, P)$ tenemos el vector aleatorio $\mathbf{x} = (x_1, \dots x_{48})$ donde cada variable aleatoria $x_i \colon \Omega \to \mathbb{R}$, $i \in \{1, \dots, 48\}$, mide una señal eléctrica del dispositivo (rango $\mathbb{R}$ pues no se nos especifica ninguna cota), y la variable aleatoria $y \colon \Omega \to \{1, \dots, 11\}$ que clasifica el estado del motor. Por lo que $\mathcal{X} = \mathbf{x}(\Omega) = \mathbb{R} \times \overset{48}{\cdots} \times \mathbb{R} = \mathbb{R}^{48}$ y $\mathcal{Y} = y(\Omega) = \{1, \dots, 11\}$

A partir del fichero alojado en Dataset for Sensorless Drive Diagnosis se toma $N < 58509$, que será el tamaño de la muestra de entrenamiento. Reservando el 20 % de los datos para test obtuvimos $N = 46807$

Ahora podemos distinguir dos casos:

- Asumir que existe $f \colon \mathcal{X} \to \mathcal{Y}$ determinista (determina la clase de cada $w \in \Omega$, configuración de componentes intactos y defectuosos, a partir de las características obtenidas con $\mathbf{x}(w)$).
- No asumir esa función determinista y tomar como función objetivo

  $$f(\mathbf{x}) = (P(y = 1|\mathbf{x}), \dots, P(y = 11|\mathbf{x}))^T$$

  para luego, en base a la Regla de Bayes, asignar cada $\mathbf{x}$ a la clase más probable. Es el caso de Regresión Logística multiclase, que será nuestro modelo de primera elección.

Nota: nos limitamos a modelos $\mathcal{H}$-lineales.

En ambos casos, para conseguir estimación g $\mathcal{H}$-lineal de f, y dado que tenemos muestra i.i.d., se seguirá el criterio ERM (minimización de riesgo empírico). Como tenemos que cuidar la cota de error de generalización, aplicaremos regularización para evitar sobreajuste.

2.2. Clases de funciones a usar.

Realizaremos una transformación de segundo orden polinomial a los vectores de características, pues aumenta la flexibilidad de nuestra frontera de decisión. Para no aumentar en exceso la complejidad de la clase de funciones (mayor dimensión $\Rightarrow$ mayor complejidad $\Rightarrow$ mayor cota de error de generalización) reduciremos previamente la dimensionalidad de nuestros vectores de características (ver parte de preprocesado). No utilizamos transformación polinómica de mayor orden pues cuanto mayor sea la longitud de los vectores de características mayores posibilidades de disminuir el error en la muestra pero mayores posiblidades de aumentar error en la población (sobreajuste); a parte de incrementar el coste computacional.

Por lo que, como hemos comentado, aplicaremos a $\mathcal{X}$ la transformación Φ_2, que genera combinaciones polinómicas de grado menor o igual a 2 de las características

$$\Phi_2(\mathbf{x}) = (1, x_1, \dots, x_{\hat{d}}, \underbrace{x_1x_2, \dots, x_1x_{\hat{d}}, x_2x_3, \dots, x_2x_{\hat{d}}, \dots x_{\hat{d}-1}x_{\hat{d}}}_{\text{combinaciones } x_ix_j \text{ con } i<j, \quad i,j\in\{1,\dots,\hat{d}\}}, x_1^2, \dots x_{\hat{d}}^2)^T$$

Nota: aquí $\hat{d} < d = 48$ pues no utilizaremos todas las características. Podemos ver que $\Phi_2(\mathbf{x})$ tiene $1 + \hat{d} + (\sum_{i=1}^{\hat{d}-1} \hat{d} - i) + \hat{d} = 1 + 2\hat{d} + (\hat{d} - 1)\hat{d} - \frac{(\hat{d}-1)\hat{d}}{2} = 1 + 2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2}$ componentes

Dentro de nuestros modelos utilizaremos enfoque probabilístico y determinístico. A continuación se exponen dos técnicas de minimización ya conocidas para enfoque probabilístico y determinístico respectivamente, junto con la clase de funciones hipótesis en cada caso.

Nota: para `RidgeClassifier` se busca solución analítica de la transformación de nuestro problema de clasificación a uno de regresión, y tendremos $\mathcal{H} := \{h_w \colon \mathbb{R}^{2\hat{d}+\frac{(\hat{d}-1)\hat{d}}{2}+1} \to \mathbb{R} \ : \ h_w(\Phi_2(\mathbf{x})) = w^T\Phi_2(\mathbf{x}),\ w \in \mathbb{R}^{2\hat{d}+\frac{(\hat{d}-1)\hat{d}}{2}+1}\}$

Regresión Logística multiclase (One-vs-Rest):

Nuestra primera elección es utilizar Regresión Logística multiclase pues en situaciones reales tiene más sentido pensar situaciones probabilísticas que en funciones deterministas. Además la función de error será fácilmente minimizable mediante Gradiente Descendente Estocástico. Asignaremos cada vector de características a la clase que se estime más probable.

Si

- Tomamos como función objetivo $f \colon \mathcal{X} \to [0,1]^{11}$ con $f(\mathbf{x}) = (P(y=1|\mathbf{x}), \dots, P(y=11|\mathbf{x}))^T$ la función que asigna a cada vector de características el vector de probabilidades de pertenecer a cada clase
- Denotamos con d a $2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2}$
- Denotamos con w_i, $i = 1, \dots, K$, con $K = 11$, el hiperplano que separa la clase i del resto

tenemos la clase de funciones

$$\mathcal{H} := \Big\{ h_{w_1,\dots w_K} \colon \mathbb{R}^{d+1} \to \mathbb{R}^K \ : \\ h_{w_1,\dots w_K}(\Phi_2(x)) = Softmax(\ (w_1^T\Phi_2(x), \dots, w_K^T\Phi_2(x))^T\ ),\ w_1, \dots w_K \in \mathbb{R}^{d+1} \Big\}$$

esto es

$$\mathcal{H} := \Big\{ h_{w_1,\dots w_K} \colon \mathbb{R}^{d+1} \to \mathbb{R}^K \ : \\ h_{w_1,\dots w_K}(\Phi_2(x)) = \frac{1}{\sum_{k=1}^K e^{w_k^T\Phi_2(x)}}(e^{w_1^T\Phi_2(x)}, \dots, e^{w_K^T\Phi_2(x)})^T,\ w_1, \dots w_K \in \mathbb{R}^{d+1} \Big\}$$

El error dentro de la muestra (máxima verosimilitud) es

$$E_{in}(w_1, \dots, w_K) := -\ln L(Y|w_1, \dots, w_K) = -\sum_{n=1}^N \sum_{k=1}^K y_{nk} \ln \sigma(w_k^T x_n)$$

donde σ es la función logística (sigmoide), $\sigma \colon \mathbb{R} \to [0,1]$, $\sigma(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{e^x+1}$
Vemos que

$$\nabla_{w_j} E_{in}(w_1, \dots, w_K) = \sum_{n=1}^N (\sigma(w_j^T x_n) - y_{nj}) x_n$$

Utilizaremos SGD para minimizar (técnica ya explicada en el segundo aparatado del problema de regresión). Una vez acabada la optimización, obtendremos los $w_1, \dots, w_K$ que minimizan E_{in}. Asignaremos entonces cada x a la clase más probable, es decir a la clase $j \in \{1, \dots, K\}$ donde

$$j = \arg max_{j \in \{1,\dots,K\}} \ \frac{exp(w_j^T x)}{\sum_{k=1}^K exp(w_k^T x)}$$

PLA-Pocket:

Recordamos que el *Algoritmo de Aprendizaje Perceptron* es caso particular del algoritmo *Gradiente Descendente Estocástico* para tamaño minibatch 1 y tasa de aprendizaje 1 para la función

$error(w^T x_n, y_n) = \text{máx}\{0, -y_n w^T x_n\}$, las funciones hipótesis tiene la forma $h_w(x) = sign(w^T x)$ donde x tiene 1 en la primera componente. La regla de adapación del algoritmo es

$$\begin{cases} w_{updated} = w_{current} + y_i x_i & sign(w^T x_i) \neq y_i \\ w_{updated} = w_{current} & sign(w^T x_i) = y_i \end{cases} \qquad (*)$$

es decir, teniendo un dataset $\mathcal{D}$ de tamaño N, $\mathcal{D} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$, lo que hacemos es recorrerlo con $i = 1, \ldots, N$ y actuar conforme (*) para cada i. Por lo que, si x_i no está bien clasificado, entonces actualizamos el vector de pesos actual, w, en la dirección correcta; si está bien casificado no se hace nada. Estos recorridos del dataset siguiendo este criterio los repetimos hasta que se consiga un recorrido completo del dataset en el que no haya ningún punto mal clasificado. *Nota:* si la muestra no es separable el algoritmo no llegará a clasificar correctamente toda la muestra, en este caso debe añadirse otra condición de parada, número máximo de épocas por ejemplo.

PLA-Pocket equivale a PLA con dos distinciones: (1) no hay restricción de seguir mientras haya puntos mal clasificados, pues la idea es utilizarlo en muestras no separables, y (2) iremos anotando la mejor solución obtenida hasta el momento, la que proporcione menor error en la muestra. Al completar las épocas máximas indicadas se devolverá el vector de pesos óptimo.
En nuestro caso, de nuevo denotando con d a $2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2}$, tenemos la clase de funciones

$$\mathcal{H} := \{h_w \colon \mathbb{R}^{d+1} \to \mathbb{R} \ : \ h_w(\Phi_2(x)) = sign(w^T \Phi_2(x)), \ w \in \mathbb{R}^{d+1}\}$$

La función de error a minimizar es

$$E_{in}(h_w) := \frac{1}{N} \sum_{n=1}^{N} [[h_w(\Phi_2(x)) \neq y_n]]$$

Modelo por regresión:
El último de nuestros modelos consiste en transformar el problema de clasificiación a uno de regresión y tendremos $\mathcal{H} := \{h_w \colon \mathbb{R}^{2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2} + 1} \to \mathbb{R} \ : \ h_w(\Phi_2(\mathbf{x})) = w^T \Phi_2(\mathbf{x}), \ w \in \mathbb{R}^{2\hat{d} + \frac{(\hat{d}-1)\hat{d}}{2} + 1}\}$, $E_{in}(w) = \frac{1}{N} \|Xw - y\|^2$

2.3. Hipótesis finales que se usarán.

En el preprocesado: (1) se eliminan características con coeficiente de correlación lineal en valor absoluto mayor a 0.95, (2) se eliminan las características con varianza 0, (3) se seleccionan las 20 características con mejor resultado por test ANOVA, (4) se realiza transformación polinomial de segundo orden a cada vector de características, (5) se normalizan las características (media 0 y varianza 1).

La métrica a valorar es accuracy. Evaluaremos desempeño de métodos `Perceptron` (PLA-Pocket), `OneVsRestClassifier` junto con `SGDClassifier` (Regresión Logística multiclase One-vs-Rest (SGD)) y `RidgeClassifier` (solución analítica de regresión por SVD a la transformación del problema de clasificación a uno de regresión). Para cada uno evaluaremos mediante 5-fold cross validation los parámetros de regularización 0.00001, 0.001 y 0.1 para regularización Ridge

2.4. Generación de conjuntos training y test.

Hay un compromiso en la selección del tamaño de entrenamiento y test: el tamaño de la muestra de entrenamiento determina el número de instancias que tendremos para entrenar el modelo y por tanto para minimizar E_{in}, y el tamaño del conjunto de test condicionará la estimación, E_{test}, de E_{out} que obtengamos. Evitaremos proporciones extremas para tamaño de test. Suele ser habitual reservar un 20 % de las instancias para test, y eso hemos hecho. De las 58509 instancias tendremos 11702 para test, el resto para training y validación cruzada. La división en training y test se ha

hecho conservando la proporción de clases, esto se hace para que tanto training como test sean los más representativos posibles de la distribución de la que provienen.

Utilizaremos k-fold cross validation con $k = 5$ para la selección de modelos. Esta técnica es derivada de *Leave-one-out*, la cual nos proporciona una estrategia para abordar la problemática o compromiso de elección del tamaño de conjunto de validación, K, esta es: $E_{out}(g) \approx E_{out}(g^-)$ cuando K pequeño y $E_{out}(g^-) \approx E_{val}(g^-)$ cuando K grande.

La técnica k-fold cross validation consiste en dividir la muestra de entrenamiento en k particiones, cada una de tamaño aproximado $\frac{N}{k}$, e ir iterando sobre ellas eligiendo cada vez una partición para actuar como conjunto de validación y entrenando sobre el resto de la muestra el modelo; una vez iteradas sobre todas las particiones se toma la media de errores obtenidos y ese es el error de validación cruzada E_{cv} que es un buen estimador de E_{out} cuando k es grande, cuanto menor sea k peor será la estimación de E_{out}. Se elige el modelo con menor E_{cv}

Se podría tomar $k = N$ pero esto implica un alto coste computacional; es por esto que se elige $k << N$, en nuestro caso $k = 5$ (es habitual $5 \leq k \leq 10$)

2.5. Detalles del preprocesado de datos.

Pasos realizados:

1. Se eliminan las características que tienen valor absoluto de coeficiente de correlación lineal mayor a 0.95 con otra característica.

   Esto se hace porque se observó la matriz de coeficientes de correlación lineal en valores absolutos

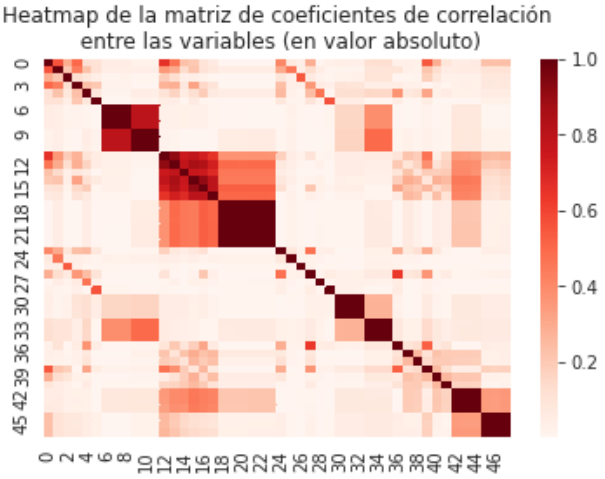

Figura 3: Matriz de coeficientes de correlación en valor absoluto

   Las características $7, 8, 10, 11, 19, 20, 21, 22, 23, 31, 32, 34, 35, 43, 44, 46, 47$ tenían valores de coeficiente de correlación mayor a 0.95 en la matriz. Se puede decir que estas características no nos aportan información nueva, además aumentan la dimensionalidad y por tanto empeoran la cota de error de generalización. Se decide eliminarlas y reducir así la complejidad de la clase de funciones. Tras eliminarlas se obtiene

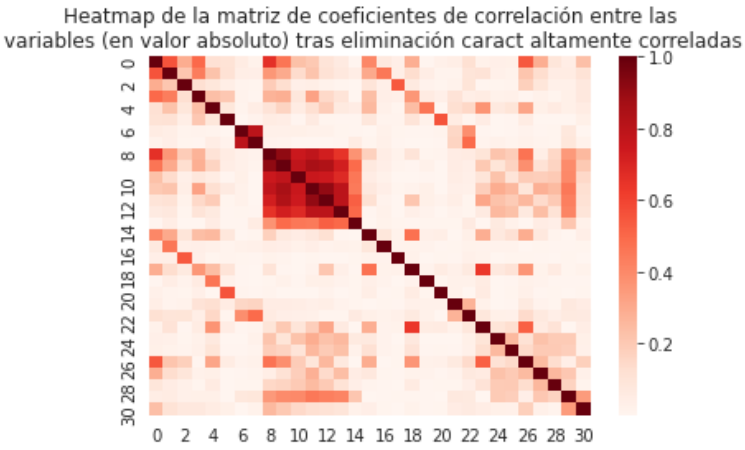

Figura 4: Matriz de coeficientes de correlación en valor absoluto tras la eliminación de características altamente correladas linealmente

una matriz de menor dimensión y con menor apreciación de zonas correspondientes a alta correlación (excepto en diagonal donde tenemos coeficientes de correlación 1 pues es la correlación de cada característica consigo misma).

2. Eliminamos las características con varianza 0 si las hubiese, no aportan ninguna información y además es necesario eliminarlas para después normalizar.

3. Seleccionamos las 20 características con mejor resultado (posible mayor "impacto" en etiquetado) en test ANOVA, el número elegido es totalmente arbitrario, se decidió elegir una cantidad algo mayor a la mitad de las características; suficiente para no quedarnos con pocas características y perder información, pero tampoco excesiva como para no poder manejar el coste computacional del aumento de dimensionalidad tras la transformación polinómica (con 20 características ya aumentaremos hasta 231 características).
   La elección de este método ha sido arbitraria, se podría haber elegido cualquier otro método para reducir la dimensionalidad y complejidad de la clase de funciones.

4. Realizamos transformación polinomial de segundo orden a cada característica para aumentar así la flexibilidad de la frontera de decisión nuestro modelo.

5. Normalizamos las características (tendrán media 0 y varianza 1), no normalizar puede degradar el desempeño de algoritmos como Perceptron y Regresión Logística mediante SGD. También deben estar normalizadas de cara a que nuestra regularización actúe correctamente. Normalizamos para evitar estas problemáticas y que todas las características "sean valoradas en la misma magnitud".

2.6. Métrica de error a usar.

Visualizaremos el número de vectores de características de la realización muestral pertenecientes a cada clase, esto nos mostrará si hay clases desbalanceadas y qué métrica necesitamos utilizar.

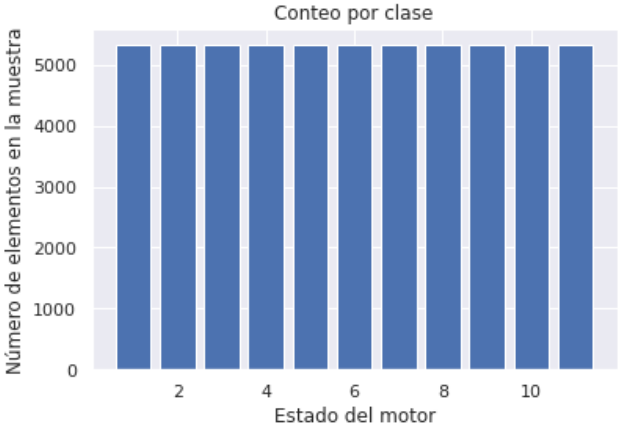

Figura 5: Número de elementos por estado de motor

Como podemos ver las clases están balanceadas, de hecho el número de elementos por clase no cambia, tenemos 5319 elementos en todas las clases. Como no tenemos clases desbalanceadas utilizaremos la métrica *accuracy* que no es más que la proporción de predicciones de clase correctas.

2.7. Parámetros usados y tipo de regularización elegida.

La regularización nos ayudará a evitar sobreajuste, sobretodo después de la transformación polinómica que hemos realizado. Con la regularización se aumentará ligeramente el sesgo para decrementar significativamente la varianza. Utilizaremos regularización Ridge, ahora tendremos error aumentado

$$E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2 = E_{in}(\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}$$

donde $\lambda \geq 0$ es el parámetro de regularización. Este tipo de regularización penaliza coeficientes de $\mathbf{w}$ grandes. Otro tipo de regularización muy conocida es regularización Lasso (utiliza norma $\|\cdot\|_1$ en lugar de norma $\|\cdot\|_2$), ésta sin embargo tiende a hacer coeficientes de $\mathbf{w}$ a 0, es buena para selección de características. En nuestro caso todas las características son de la misma naturaleza (señales eléctricas) y confiamos en la selección de características relevantes que hicimos, es por esto que preferimos regularización Ridge antes que Lasso.

En Regresión Logística hemos utilizado la función `OneVsRestClassifier` junto con `SGDClassifier`, sigue el criterio one vs rest que es el explicado la sección *Clase de funciones a usar* y el que utilizaremos. `SGDClassifier` utiliza tamaño de minibatch 1. Es interesante notar que la función de error junto con el término de regularización Ridge es convexa; pierde relevancia el uso de minibatches o punto de inicio al no tener óptimos locales. Se podría utilizar Gradiente Descendente sin ningún problema. Aún así, dado que `sklearn` nos proporciona método para la versión estocástica, y que realmente ganamos eficiencia computacional al calcular gradiente en un solo punto, y el ruido que supone computar el gradiente en un solo punto termina promediándose en número alto de iteraciones; se decide utilizar la versión que nos facilita `sklearn`. El learning rate se ha elegido adaptativo, cuando no se esté mejorando en el criterio de tolerancia tras `n_iter_no_change` épocas, entonces actualizamos la tasa de aprendizaje mediante $learning\_rate = \frac{learning\_rate}{5}$, vamos disminuyendo conforme nos acercamos al mínimo para prevenir oscilación. Se ha elegido learning rate inicial algo elevado puesto que es adaptativo y una tolerancia arbitraria `tol` 0.001 a la que daremos hasta 5 épocas para que se halla disminuido en `tol` el error. El algoritmo parará si tenemos learning rate menor a 1e-6 o por iteraciones máximas. Se adjunta código con comentarios del resto de parámetros:

```
{"model": [OneVsRestClassifier(SGDClassifier(loss = 'log', # función de pérdida
  de regresión logística
                                penalty = 'l2', # utilizaremos regularización l2
```

```
                                # alpha  constante del término de regularización (probaremos distintos valores mediante 5-fold cross validation)
                                fit_intercept = True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                                max_iter = 80, # Número máximo de iteraciones arbitrario
                                tol = 0.001, # Tolerancia para criterio de parada por tolerancia (parar si loss > best_loss - tol tras n_iter_no_change épocas seguidas)
                                                # el criterio valora si el error es no mejora en 0.001 el mejor error hasta el momento. En nuestro caso esta condición por tolerancia se usará para learning_rate adaptativo
                                shuffle = True, # mezclamos después de cada época
                                n_jobs = -1, # máxima paralelización posible en ejecución
                                random_state = 1, # para tener reproducibilidad de los resultados
                                learning_rate = 'adaptive', # si no se mejora resultado por criterio tolerancia (loss > best_loss - tol) tras n_iter_no_change épocas seguidas, entonces cambiamos learning rate por ( learning rate)/5
                                                                # queremos evitar oscilación
                                eta0 = 0.05, # learning rate inicial arbitrario
                                early_stopping = False, # False pues no queremos reservar más datos para validación
                                n_iter_no_change = 5, # cada 5 épocas sin mejora en crit. tolerancia se realizará la adaptación del learning rate ( learning_rate='adaptive')
                                class_weight = None, # se iterpreta que todas las clases tienen peso 1, que es el caso
                                average = False, # no nos interesa obtener media de pesos
                                verbose = 0, # no nos interesan mensajes
                                warm_start = False, # no reutilizamos ninguna solución anterior durante la validación cruzada
                                l1_ratio = 0 # 0 corresponde a l2 penalty, no se usará pues solo se usa si learning_rate = 'elasticnet'
                                ), n_jobs= -1)], # máxima paralelización posible
      "model__estimator__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados


```

Listing 4: Parámetros usados en SGDClassifier

Para PLA-Pocket se utiliza la función `Perceptron`, se adjunta código con comentarios sobre los parámetros elegidos:

```
{"model": [Perceptron(penalty = 'l2', # utilizaremos regularización l2
                        # alpha  constante del término de regularización ( probaremos distintos valores mediante 5-fold cross validation)
                        # l1_ratio solo si usa si penalty='elasticnet' que no es el caso
                        fit_intercept = True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                        max_iter = 80, # Número máximo de iteraciones arbitrario
                        #tol  por defecto, arbritraria, si tenemos loss > previous_loss - tol paramos
                        shuffle = False, # no mezclamos la muestra tras cada época
                        eta0 = 1, # =1 para no multiplicar las actualizaciones, no las alteramos
```

```
                            n_jobs = -1, # máxima paralelización posible en ejecución
                            random_state = 1, # para tener reproducibilidad de los resultados
                            early_stopping = False, # no nos interesa reservar más datos para validación (nuestro único criterio de parada serán las iteraciones)
                            # validation_fraction no será usado pues early_stopping = False
                            # n_iter_no_change no será usado pues early_stopping = False
                            class_weight = None, # None se interpreta como que todas las clases tienen peso 1, que es el caso
                            warm_start = False # no reutilizamos ninguna solución anterior durante la validación cruzada
                            )],
      "model__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados

```

Listing 5: Parámetros usados en Perceptron

Por último utilizaremos `RidgeClassifier`, que transforma el problema de clasificación en uno de regresión y resuelve el de regresión (en nuestro caso lo resolveremos de forma analítica mediante descomposición en valores singulares), a pesar de esto no suele dar malos resultados. Adjuntamos código con comentarios sobre los parámetros elegidos:

```
{"model": [RidgeClassifier(# alpha  constante del término de regularización (probaremos distintos valores mediante 5-fold cross validation)
                        fit_intercept=True, # añadimos sesgo o intercept pues nuestra matriz aún no tiene columna de 1s
                        normalize=False, # no nos interesa normalizar, ya lo hicimos en preprocesado
                        copy_X=True, # no nos interesa sobreescribir X
                        max_iter=None, # None pues no usaremos método iterativo
                        tol=0.001, # precisión de la solución, lo dejamos por defecto, no nos importa pues utilizaremos solución analítica por SVD
                        class_weight=None, # None se interpreta como que todas las clases tienen peso 1, que es el caso
                        solver='svd', # resolveremos de forma analítica usando descomposición en valores singulares
                        random_state=None)], # no nos hace falta para tener reproducibilidad de los resultados pues no se va a usar solver=sag o solver=saga. Obtendremos solución analítica
      "model__alpha": [0.00001, 0.001, 0.1]}, # probamos valores de regularización arbitrarios dentro de los recomendados

```

Listing 6: Parámetros usados en `RidgeClassifier`

Nota en general: el número de iteraciones, `n_iter_no_change` y tolerancia se han elegido de forma arbitraria y comprobando que hubiese convergencia. Estos parámetros se podrían haber añadido como parámetros a valorar y expandir nuestro número de modelos para validación cruzada, pero los tiempos de ejecución serían inabarcables.

2.8. Selección de la mejor hipótesis y E_{out}

Para la selección de la mejor hipótesis o modelo utilizaremos 5-fold cross validation, técnica ya explicada en la sección de *Generación de conjuntos training y test*. Elegiremos como modelo ganador aquel con menor E_{cv}. El modelo ganador se entrenará finalmente sobre toda la muestra.

Para hacer esto se ha utilizado `GridSearchCV`, función que nos permite entrenar el modelo ganador en toda muestra mediante `refit=True`, y nos facilita la visualización de resultados. La métrica considerada para elección del mejor modelo es el *accuracy* (tenemos clases perfectamente balanceadas)

Vamos a valorar distintos parámetros de regularización: tanto para Regresión Logística multiclase (`OneVsRestClassifier` con `SGDClassifier`) como para PLA-Pocket (`Perceptron`) y solución por regresión (`RidgeClassifier`) tendremos los parámetros de regularización 0.00001, 0.001 y 0.1, por lo que en total tendremos 9 modelos para 5-fold cross validation (se podrían tener más modelos considerando otros parámetros, nos hemos limitado a los parámetros de regularización indicados)

Nos referiremos a los modelos o hipótesis determinados en última instancia por `OneVsRestClassifier` con `SGDClassifier`, `Perceptron` y `RidgeClassifier` (junto sus parámetros) mediante M_{RL}, $M_{PLA-Pocket}$, M_{Regre} respectivamente (representan todos los pasos realizados: preprocesado, clase de funciones elegida, parámetros...). Tendremos 9 modelos determinados finalmente por los parámetros de regularización:

$$\{(M_{PLA-Pocket}, \lambda), (M_{RL}, \lambda), (M_{Regre}, \lambda)\} \quad \text{con} \quad \lambda \in \{0.00001, 0.001, 0.1\}$$

Veamos los accuracies medios de cada modelo obtenidos por 5-fold cross validation

Modelo	Accuracy medio 5-fold cross validation
$(M_{RL}, 0.00001)$	0.94853335
$(M_{RL}, 0.001)$	0.89749403
$(M_{RL}, 0.1)$	0.69619929
$(M_{PLA-Pocket}, 0.00001)$	0.7828322
$(M_{PLA-Pocket}, 0.001)$	0.4353621
$(M_{PLA-Pocket}, 0.1)$	0.19939233
$(M_{Regre}, 0.00001)$	0.78894203
$(M_{Regre}, 0.001)$	0.7891984
$(M_{Regre}, 0.1)$	0.78590836

Cuadro 2: Accuracies medios por 5-fold cross validation de cada modelo

Nuestro mejor modelo con accuracy medio por 5-fold cross validation 0.94853335 es $(M_{RL}, 0.00001)$, podemos ver que valores mayores de regularización nos dan peores resultados (menor accuracy). Desde luego es destacable la gran mejora en accuracy obtenida con un pequeño valor de regularización; al aumentar valores de λ estamos imponiendo restricción cada vez más fuerte y el modelo deja de ajustar bien como podemos observar, para (M_{RL}, λ) con $\lambda = 0.1$ obtenemos accuracy 0.69619929, bastante peor.

Para $M_{PLA-Pocket}$ el máximo accuracy también se obtiene con el menor valor de regularización, aunque se obtiene accuracy 0.7828322, considerablemente peor que 0.94853335, al igual que antes valores mayores de λ empeoran los resultados, en este caso aún más notable: $(M_{PLA-Pocket}, \lambda)$ con $\lambda = 0.1$ nos da accuracy 0.19939233

Finalmente podemos observar el mismo fenómeno de peores resultados al aumentar el término de regularización en M_{Regre}, y sorprendentemente vemos que hemos obtenido, con cualquier término de regularización, accuracies ligeramente superiores al mejor obtenido por $M_{PLA-Pocket}$, vemos que, a pesar de transformar el problema de clasificación a uno de regresión y utilizar una función de pérdida de error cuadrático medio, obtenemos buenos resultados.

Entrenamos nuestro modelo ganador $(M_{RL}, 0.00001)$ con todos los datos de entrenamiento. Al utilizar todos los datos de entrenamiento tenemos la ventaja de tener un tamaño de entrenamiento

mayor (en 5-fold cross validation no utilizábamos N instancias, siempre reservábamos una de las 5 particiones para validar). Entrenando sobre toda la muestra de entrenamiento se espera ahora obtener un "mejor regresor".

Utilizaremos nuestro conjunto de test que guardamos desde el principio para estimar el error de generalización, estimaremos E_{out} mediante E_{test}. Procediendo así, y teniendo en cuenta que estamos valorando mediante la métrica accuracy, estimamos error, de nuestro mejor modelo $(M_{RL}, 0.00001)$, fuera de la muestra: 1-0.94966672=0.05033328; estimación ligeramente más optimista que la que obteníamos en validación cruzada ($0.94966672 > 0.94853335$), lo cual va en consonancia con el resultado teórico $E_{out}(g) \leq E_{cv}$ (va en consonancia en el sentido de que $E_{out}(g) \approx E_{test}(g) \leq E_{cv}$), que nos viene a decir que el error de validación cruzada, E_{cv}, nos da una estimación pesimista del error de generalización.

2.9. E_{out} para la mejor hipótesis usando todos los datos para entrenar.

Ahora utilizamos todos los datos disponibles para entrenar nuestro mejor modelo, esto nos dará un mejor ajuste. Para estimar E_{out} ya no disponemos de conjunto de test. Lo que se hará es hacer uso de la cota pesimista de que nos proporciona E_{cv}, tendremos una cota superior de $E_{out}(g)$. La cota será más ajustada cuanto mayor sea k en k-fold cross validation, idealmente $k = N$ (Leave-one-out); de nuevo, por el alto coste computacional que esto supondría, nos conformaremos con $k = 20$

Procediendo así, y teniendo en cuenta que estamos valorando mediente la métrica accuracy, estimamos $E_{out}(g) \leq 1 - 0.950024741340530 9 = 0.04997525865946906$. Es destacable que al estar usando todos los datos disponibles para entrenar estamos obteniendo un "mejor clasificador", estamos obteniendo como cota pesimista de E_{out} un valor menor que la estimación de E_{out} por E_{test} cuando solo usábamos el conjunto de training para entrenar (que a su vez era menor que el error de validación cruzada).