Práctica 3

Ajuste avanzado de modelos lineales

Miguel Ángel Fernández Gutiérrez

Aprendizaje Automático
4º Doble Grado en Ingeniería Informática y Matemáticas
Universidad de Granada

mianfg.me

Índice

1 Introducción

El objetivo de esta tercera práctica es ajustar el modelo lineal más adecuado posible para dos conjuntos de datos, correspondientes a problemas de clasificación y regresión.

La estrategia que seguiremos en los dos problemas será la siguiente:

1. Comprender el problema que pretendemos resolver.
2. Preprocesar los datos para poder trabajar adecuadamente con ellos.
3. Elegir una serie de clases de hipótesis lineales.
4. Fijar un conjunto de modelos y usar validación para escoger el mejor de ellos.
5. Analizar los resultados y estimar el error del modelo.
6. Sacar conclusiones del desempeño de nuestro modelo.

Para la implementación haremos uso principalmente de las funciones de `scikit-learn`, que se especificarán en esta memoria. El código, a su vez, está dividido en tres archivos: uno para el problema de regresión (`P3_regresion.py`), otro para el de clasificación (`P3_clasificacion.py`) y uno con funciones auxiliares de uso común (`P3_common.py`).

2 | Problema de regresión: *superconductivity*

2.1. Descripción del problema

Para el problema de regresión, usaremos un conjunto de 21263 instancias que recogen información de 82 atributos de superconductores, así como la propiedad que queremos predecir: su *temperatura crítica*, la temperatura a partir de la cual el material se comporta como conductor. También se nos proporciona un set de datos que contiene la fórmula química para cada superconductor. Para entrenar nuestro modelo sólo usaremos los primeros (los que tienen 82 atributos). Estos datos pertenecen a SuperCon (Superconducting Material Database).[1]

Los datos están dispuestos en filas, correspondiendo las primeras 81 columnas a atributos de cada material (número de elementos, masa atómica media, etc.), y la última de ellas a la temperatura crítica. Tenemos por tanto un problema de **regresión** en el que $\mathcal{X} = \mathbb{R}^{81}$, $\mathcal{Y} = [0, \infty)$ y queremos aprender $f : \mathcal{X} \longrightarrow \mathcal{Y}$ (desonocida) que, a cada conjunto de atributos del superconductor, asigna la temperatura crítica.

Para poder profundizar en nuestros datos, procederemos a visualizarlos. Para ello seguiremos dos estrategias:

- Seleccionamos las variables con mayor importancia y hacemos plots en 2D de estas variables. Hay muchas formas de hacerlo, escogeremos el criterio de `DecisionTreeRegressor` (árboles de decisión).[2]
- Un histograma del atributo que queremos predecir, para ver cómo se distribuye.

Para la primera estrategia, usamos la librería de `scikit-learn` correspondiente, obteniendo las importancias que podemos ver a continuación:

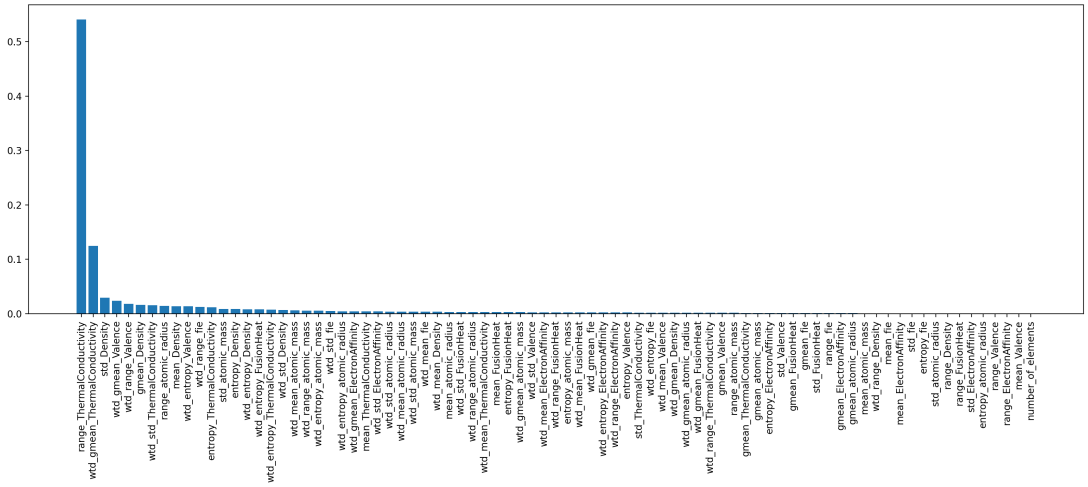

Figura 1: Importancia de variables según `DecisionTreeRegressor`

[1] *Superconducting Material Database* (SuperCon), National Institute for Materials Science. https://supercon.nims.go.jp/en/

[2] How to Calculate Feature Importance With Python, *Machine Learning Mastery*. https://machinelearningmastery.com/calculate-feature-importance-with-python/

Vemos que las variables que más poder predictivo tienen, con diferencia, son `range_ThermalConductivity` y `wtd_gmean_ThermalConductivity`, en ese orden. Representaremos estos atributos junto con la variable a predecir para nuestros datos.

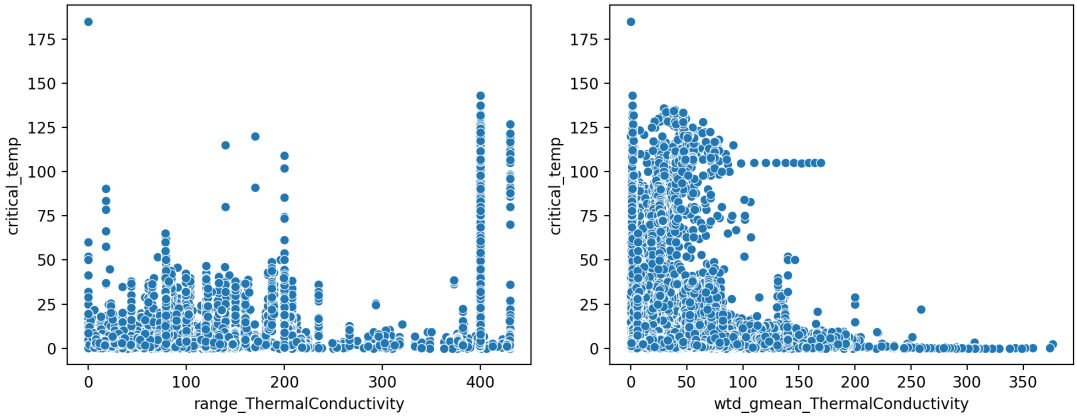

Figura 2: Relación de variables más significativas con el *target*

Desafortunadamente, vemos que para estos atributos es complicado intuir una regla de regresión que los relacione con nuestro objetivo. Veremos ahora cómo se distribuye la variable que queremos predecir, mediante un histograma de ésta.

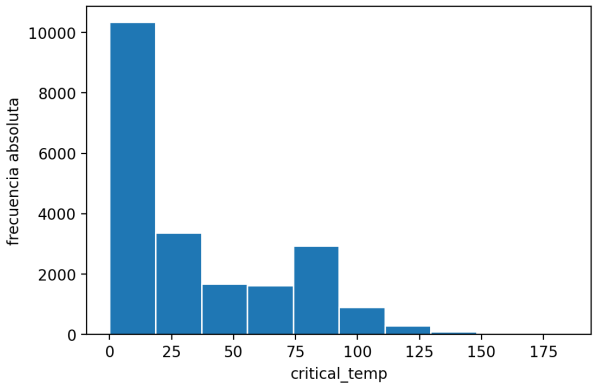

Figura 3: Histograma de atributo *target*

Podemos ver que, en la mayoría de ejemplos, las temperaturas críticas son bajas, siendo una temperatura crítica alta algo más bien raro.

2.2. Selección de $\mathcal{H}$

Tras la exploración de los datos efectuada, vemos que es difícil que los datos se distribuyan linealmente. Es por ello por lo que consideraremos transformaciones no lineales del espacio, en concreto, combinaciones cuadráticas. Mediante esto, haremos que nuestro modelo sea más sofisticado, esperando aproximar los datos mejor, esperando que sigan una distribución lineal en el espacio transformado.

Es importante comentar llegado a este punto que al seleccionar una transformación polinómica para $\mathcal{H}$ estamos aumentando la complejidad de $\mathcal{H}$. He considerado, sin embargo, interesante hacer esto por dos motivos: el primero de ellos es operativo, pues el problema tiene una complejidad que nos conduce a pensar que el ajuste cuadrático puede ser más fino; el segundo de ellos es educativo, pues podremos estudiar transformaciones no lineales en esta práctica.

Al usar este espacio transformado tenemos el riesgo de caer en *overfitting*. Sin embargo, emplearemos técnicas de regularización para evitar esto.

Pasamos a definir el espacio transformado $\mathcal{Z}$. Partiendo de la base de que haremos reducción de dimensionalidad, pasaremos de tener un espacio $\mathcal{X}$, d-dimensional, a tener un espacio X', d'-dimensional, con $d' \leq d$. A este espacio será el que le haremos las transformaciones polinómicas, mediante la función:

$$\begin{aligned} \Phi_2 : \mathcal{X}' &\longrightarrow \mathcal{Z} \\ (x_1, ..., x_{d'}) = x \mapsto \Phi_2(x) = (1, x_1, ..., x_{d'}, x_1^2, ..., x_{d'}^2, x_1x_2, x_1x_3, ..., x_1x_{d'}, \\ x_2x_3, ..., x_2x_{d'}, ..., x_{d'-1}x_{d'}) \end{aligned}$$

y obtenemos un espacio transformado $\mathcal{Z} = \Phi_2(\mathcal{X}')$, $\tilde{d}$-dimensonal, con

$$\tilde{d} = 1 + 2d' + \frac{d'(d'-1)}{2}$$

En nuestro caso, será $d' < 81$.

Una vez definido nuestro espacio transformado podemos considerar la clase de funciones para regresión:

$$\mathcal{H} = \left\{ w_0 + \sum_{i=1}^{d'} w_i x_i + \sum_{i=1}^{d'} w_i' x_i^2 + \sum_{i=1}^{d'} \sum_{j=i}^{d'} w_{ij} x_i x_j \ : \ w_i, w_i', w_{ij} \in \mathbb{R} \right\} = \left\{ w^T \Phi_2(x) \ : \ w \in \mathbb{R}^{\tilde{d}} \right\}$$

2.3. Selección de conjuntos de entrenamiento, validación y *test*

Dado que los datos proporcionados no están divididos, haremos uso de la función `train_test_split` para separarlos en un 20 % de datos de *test* y un 80 % de datos de *train*, mezclando los datos para evitar cualquier tipo de sesgo que pueda surgir de su ordenación. Además, respecto a la elección de conjuntos de validación, dado que efectuaremos *K-fold cross validation* no será necesario especificarlos.

2.4. Preprocesado

A continuación, haremos un preprocesamiento de los datos, por pasos. Resumimos los pasos de preprocesado, y cómo actúa en nuestro modelo, en la siguiente gráfica.

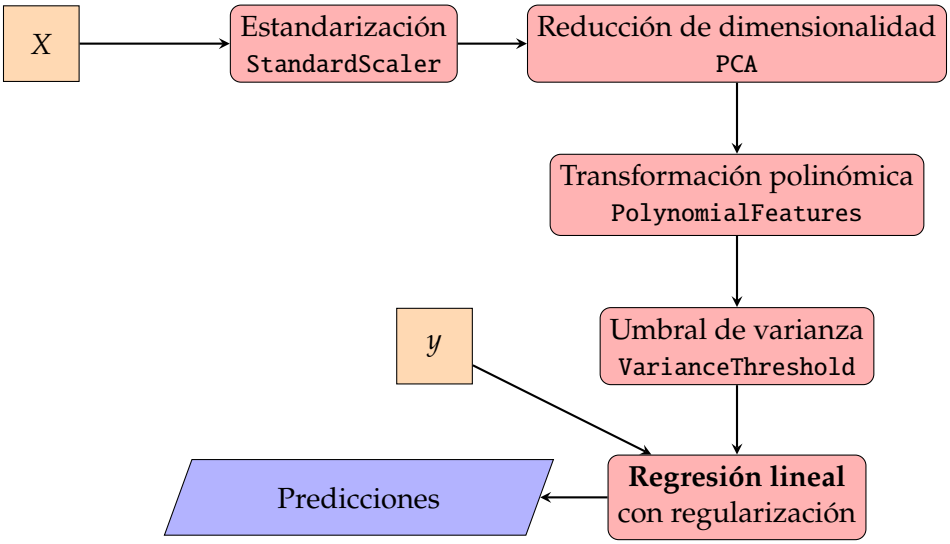

Figura 4: Diagrama de preprocesado para el problema de regresión

Todos los pasos de preprocesado serán implementados mediante el uso de *pipelines* en `scikit-learn`. En concreto, el *pipeline* de preprocesado para este problema puede verse en la variable `preprocessing`. Tras aplicar preprocesado, podemos ver cambios sustanciales en la matriz de correlación, pasando de variables fuertemente correladas a variables casi incorreladas. Esto nos indica que hemos decrementado considerablemente la redundancia de nuestras variables.

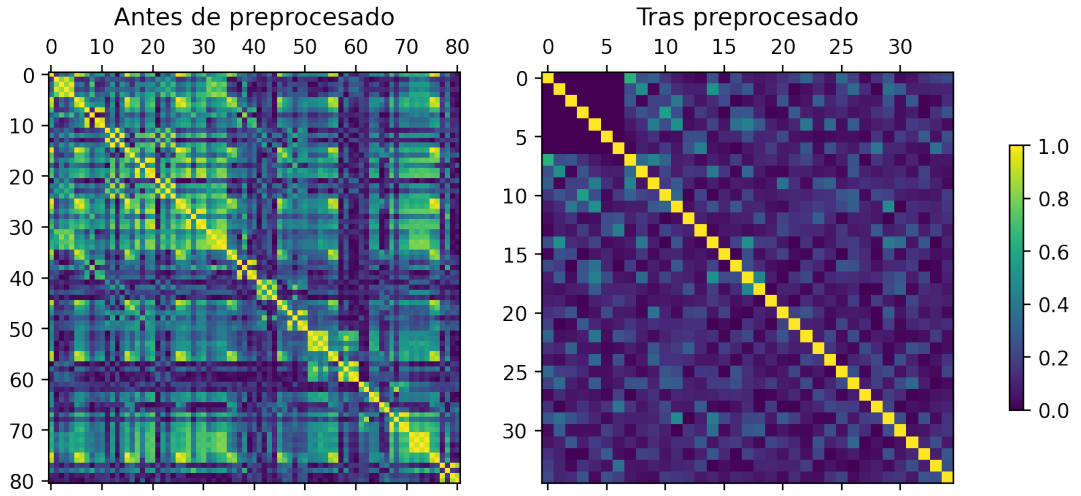

Figura 5: Matriz de correlación antes y después del preprocesado

2.4.1. Tratamiento de valores perdidos y *outliers*

En muchos conjuntos de datos es posible que algunos ejemplos no tengan algún que otro atributo, en cuyo caso sería necesario procesar los datos de una forma específica. En nuestro caso, vemos que no hay valores perdidos, por lo que no añadiremos una fase de preprocesado con el objetivo de tratar con éstos.

Por otra parte, respecto a la presencia de *outliers*, podríamos encargarnos de detectarlos y eventualmente eliminarlos, pero no lo hemos hecho dado que desconocemos si esos puntos pueden tener información valiosa para nuestro problema. Lo ideal sería consultar con un experto si es posible esta eliminación.

2.4.2. Estandardización

La estandardización es el reescalado de los datos para que tengan las propiedades de una distribución normal estándar (de media nula y desviación estándar unidad, $\mathcal{N}(0,1)$). Este paso es necesario porque muchos algoritmos requieren que las características estén normalizadas para un funcionamiento adecuado. De hecho, la estandardización es especialmente relevante en el caso de PCA. En PCA, nos interesan los componentes que maximizan la varianza. Si un componente varía menos que otro por sus escalas respectivas, es posible que PCA determine que la dirección de mayor varianza corresponde con los que más varíen, si no se escalan las varianzas.[3]

A continuación, explicaremos exhaustivamente por qué hemos decidido efectuar cada paso, así como los pasos que hemos omitido.

2.4.3. Reducción de dimensionalidad

Es esencial reducir la dimensionalidad de nuestro problema, y en especial si tratamos con un espacio transformado (el cuadrático), como haremos en nuestro caso. La reducción de dimensionalidad nos permite:

- Reducir el *overfitting*: datos menos redundantes conducen a menos oportunidad para que nuestro modelo tome decisiones basadas en el ruido.
- Reducir el tiempo de entrenamiento.
- Aumentar la precisión, ya que usamos datos menos "engañosos".

Para analizar la importancia de este paso, realizaremos una ejecución sin utilizar la reducción de dimensionalidad, y comprobaremos que obtenemos mejores resultados (junto a todas las ventajas que acabamos de comentar).

Para nuestro problema, utilizaremos el PCA (*Principal Component Analysis*), una técnica de selección no supervisada, que considera combinaciones lineales de las variables (componentes principales) que permitan recoger la mayor varianza de los datos de acuerdo a diversas proyecciones. Realizamos PCA indicando que la varianza acumulada de las variables tras la aplicación

[3]Importance of feature scaling https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html

del algoritmo debe ser un 80 % de la original. De este modo, pasamos a tener 35 variables (de las 81 variables que teníamos inicialmente).

La elección del porcentaje de varianza del PCA depende de cómo de agresiva queremos que sea la reducción: cuanto menor sea el valor, más agresiva será. Un porcentaje del 80 % es una reducción medianamente agresiva, justificada por la variabilidad que pueden tener nuestros datos (tienen un ruido considerable).

2.4.4. Transformaciones polinómicas

Como dijimos anteriormente al seleccionar $\mathcal{H}$, vamos a utilizar transformaciones polinómicas cuadráticas. La implementación en `scikit-learn` es sencilla, simplemente hacemos uso de `PolynomialFeatures(2)`.

2.4.5. Umbral de varianza

Además, como paso adicional eliminaremos las variables que tengan una varianza muy baja (menor a 0.15), pues darán poca información para resolver nuestro problema.

2.5. Medida del error

En el caso de regresión tenemos muchas métricas de error que considerar, nosotros tendremos en cuenta dos de ellas.

- **Error cuadrático medio (*mean squared error*, MSE).** Usa la distancia entre las predicciones y las etiquetas obtenidas. Dada una función $h \in \mathcal{H}$, calculamos su error de la siguiente forma:

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

  Para discutir los resultados usaremos su raíz cuadrada, pues de ese modo estaremos en la escala de nuestros datos. Una de las principales desventajas de esta métrica es que, al no estar acotada, sólo la podemos usar para comparar ajustes en un mismo conjunto de datos, y no entre conjuntos distintos. Sin embargo, nos servirá como métrica de entrenamiento.

- **Coeficiente de determinación (R^2).** Este coeficiente está acotado e indica la bondad de ajuste. Precisamente por lo primero nos puede servir para comparar entre distintos conjuntos de datos. Se define como sigue para una función $h \in \mathcal{H}$:

$$\text{R}^2(h) = 1 - \frac{\sum_{n=1}^{N}(y_n - h(x_n))^2}{\sum_{n=1}^{N}(y_n - \bar{y})^2} \qquad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_n$$

2.6. Regularización

Como hemos comentado anteriormente, usaremos regularización para evitar el *overfitting*, mediante la limitación de la complejidad del modelo. Mediante esta técnica, obtendremos una mejor generalización. Utilizaremos dos regularizadores:

- **Regularización L_2 (ridge).** Añadimos a la función de pérdida una penalización proporcional al cuadrado de los pesos:

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

  Esta regularización favorece que los coeficientes de los atributos sean bajos. Es especialmente útil cuando la mayor parte de los atributos son relevantes, algo muy interesante en nuestro caso, pues hemos preprocesado los datos precisamente para esto.

- **Regularización L_1 (lasso).** La penalización es, en este caso, lineal:

  $$L_{reg}(w) = L(w) + \lambda \sum_i w_i$$

  La regularización lasso favorece que algunos de los coeficientes valgan cero. Funciona mejor si los atributos no están correlados entre sí.

Dado que L_1 tiende a minimizar la relevancia de los atributos y L_2 la correlación (los efectos contrarios), probaremos ambos en nuestro problema. Además, hemos de considerar que $\lambda > 0$ es un *hiperparámetro* que controla la intensidad de regularización (cuanto mayor sea, más regularización). Para encontrar el valor adecuado de λ usaremos la estrategia de validación. La validación, por tanto, no nos indicará únicamente el mejor modelo para nuestros datos (como veremos más adelante), sino el mejor coeficiente de regularización.

2.7. Selección de algoritmos

Seleccionaremos los modelos para nuestro ajuste. Para ello emplearemos *K-fold cross validation*, una técnica que divide el conjunto de entrenamiento en K conjuntos con la misma cantidad de datos, y realizando K entrenamientos, de modo que en cada uno de ellos el modelo entrena los datos de los $K-1$ conjuntos y los evalúa en el conjunto sobrante. A continuación, calculamos la media de los errores a lo largo de todos los conjuntos de validación y escogemos el modelo de error menor. Este error, el error de cross-validation, E_{cv}, es un buen estimador de E_{out}, por lo que nos servirá para escoger el mejor modelo en términos de generalización y error.

Cada vez que seleccionamos un modelo, también estamos seleccionando sus hiperparámetros. De este modo, al emplear la técnica de validación, obtendremos el mejor modelo con los mejores parámetros. Además, y con el objetivo de eliminar el *data snooping*, realizaremos todas las fases de preprocesado y selección de modelos de una vez.

Para hacer todo esto, usamos los *pipelines* y la función `GridSearchCV` de `scikit-learn`. Una vez encontrado el mejor modelo, lo entrenamos sobre todo el conjunto de entrenamiento, para un mayor rendimiento.

En el caso de nuestro problema, consideraremos modelos de regresión lineal, usando la métrica MSE. Es interesante comentar que, dado que `GridSearchCV` busca maximizar la métrica, le pasaremos el opuesto de MSE (`neg_mean_squared_error`). Usaremos los siguientes modelos:

- **`SGDRegressor`**: gradiente descendiente estocástico. Usaremos todas las combinaciones de los siguientes parámetros:
  - **Función de pérdida.** Puede ser `squared_loss`, que es el ajuste por mínimos cuadrados, o `epsilon_insensitive`, que ignora los errores menores que un cierto ϵ (por defecto, $\epsilon = 0.1$).
  - ***Learning rate.*** Puede ser `adaptive`, que inicia $\eta = \eta_0$ y va dividiéndolo por 5 si alcanzamos un cierto número de iteraciones sin disminuir el error; o `optimal`, que usa $\eta = \frac{1}{\alpha(t+t_0)}$.
  - **Regularización.** Usaremos los dos tipos de regularización comentados, `l2` y `l1`.
  - **Constante de regularización.** Probaremos con 10 valores diferentes de λ, igualmente espaciados (logarítmicamente) en $[10^{-5}, 10^5]$.

  El resto de parámetros son los parámetros por defecto de `SGDRegressor`.
- **`Ridge`**: modelo lineal con regularización L_2. Usaremos diversas constantes de regularización, como en el caso de `SGDRegressor`: 10 valores diferentes de λ, igualmente espaciados (logarítmicamente) en $[10^{-5}, 10^5]$. El resto de parámetros son los parámetros por defecto.
- **`Lasso`**: modelo lineal con regularización L_1. Usamos las mismas constantes que `Ridge`.

2.8. Análisis de resultados

Tras realizar el proceso anteriormente descrito, nos disponemos a comentar los resultados. El mejor modelo obtenido ha sido `Lasso` (regresión lineal con penalización L_1), con un $\lambda = 46.415888$, una regularización muy fuerte – algo esperable dado que nuestros datos se encuentran dispersos de una forma muy compleja, por lo que será necesaria una mayor regularización para evitar el *overfitting*. Obtenemos en este modelo los siguientes resultados:

$\sqrt{\text{MSE}}_{cv}$	$\sqrt{\text{MSE}}_{in}$	$\sqrt{\text{MSE}}_{test}$	R^2_{in}	R^2_{test}	Tiempo
18.854377	18.814792	18.760032	0.699395	0.695504	203.85 s

2.8.1. Selección de la mejor hipótesis

Nos decantamos por tomar como hipótesis definitiva g este modelo, de nuevo con el fundamento teórico de que E_{cv} es buen estimador de E_{out}.

Para finalizar nuestro análisis de los resultados, recurriremos a la curva de aprendizaje de nuestro modelo. Para obtenerla, iremos entrenando con porciones incrementales del conjunto de *test*, dejando a un lado conjuntos de validación para poder evaluar en paralelo.[4]

[4] *Plotting Learning Curves*, scikit-learn. https://scikit-learn.org/stable/auto_examples/model_selection/plot_learning_curve.html

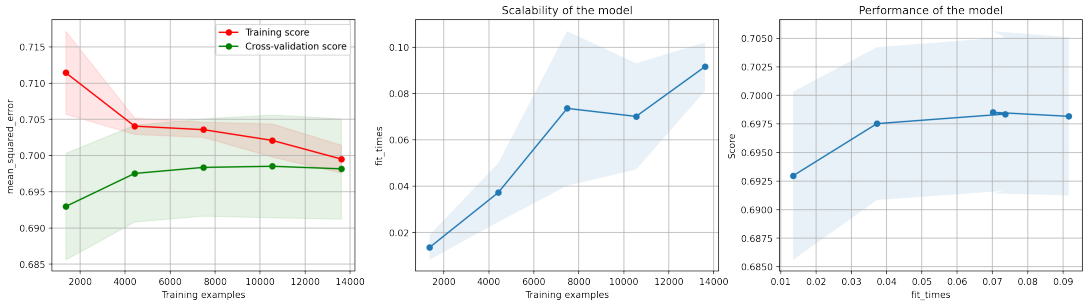

Figura 6: Curvas de aprendizaje para el problema de regresión

Tenemos a nuestra disposición tres gráficas a analizar. Comenzaremos con la más interesante, la primera de ellas, en el que se presenta la evolución de los errores de entrenamiento y de *cross-validation* conforme aumenta el número de ejemplos. El comportamiento es justamente el esperado: el error en el entrenamiento va disminuyendo conforme tenemos más ejemplos, mientras que en *cross-validation* va aumentando (al igual que lo haría E_{out}).

Las dos gráficas restantes nos indican la escalabilidad y el rendimiento del modelo. La gráfica de escalabilidad mide el número de ejemplos de entrenamiento frente a los tiempos de entrenamiento. Vemos cómo el tiempo va incrementándose como aumenta el número de datos, como es esperable. En el caso de la gráfica de rendimiento, vemos cómo la capacidad del modelo crece de forma logarítmica.

2.8.2. Estimación de E_{out}

Sabemos que el error de *cross-validation* es buen estimador del error fuera de la muestra. Sin embargo, usaremos los errores que nos proporciona el conjunto de *test*, pues son datos que no han sido analizados por nuestro modelo. Obtuvimos una raíz de MSE de 18.76 y un R^2 de 0.70, los cuales nos indican a pensar que nuestro modelo ha aprendido, aunque el valor de correlación está lejos de ser óptimo.

Respecto a la estimación de E_{out}, deberemos conformarnos con los resultados de *test*, pues desconocemos la forma de aplicar la cota de Hoeffding en este caso.

2.9. Conclusiones

Vamos a analizar la calidad del modelo que hemos obtenido. Para ello, compararemos los resultados que hemos obtenido con dos variaciones:

- Sin implementar transformaciones polinómicas (y con *cross-validation*).
- Comparando con un estimador aleatorio (sin *cross-validation*).

Tras estas dos variaciones, obtenemos los siguientes resultados, que comparamos con el modelo que hemos obtenido:

	$\sqrt{\text{MSE}}_{in}$	$\sqrt{\text{MSE}}_{test}$	R^2_{in}	R^2_{test}	Tiempo
Modelo conseguido	18.814792	18.760032	0.699395	0.695504	203.85 s
Sin polinomios	22.620530	22.498896	0.565487	0.562039	61.64 s
Dummy	34.316372	34.002158	0.000000	–0.000291	1.07 s

Observamos inicialmente cómo el regresor *dummy* da unos resultados muy malos. Sin embargo, el interés de este regresor es que, al ser aleatorio, cualquier regresor medianamente bueno debería superar a éste, que hace una estimación media. En comparación con *dummy*, nuestro modelo consigue un error un poco superior a la mitad.

Por otra parte, vemos cómo las transformaciones polinómicas tienen sentido, y es algo que podemos apreciar con más fuerza con el coeficiente R^2. Esto nos indica que nuestra selección de variables ha sido satisfactoria, y que las transformaciones polinómicas nos han permitido mejorar nuestro modelo.

Viendo los datos que hemos obtenido, podemos decir que hemos obtenido un modelo con un error aceptable. Evidentemente no es el mejor, pero podemos dar nuestro estudio por satisfactorio, al haber logrado una mejora respecto de otros modelos que acabamos de comparar.

3 Problema de clasificación: *sensorless drive diagnosis*

3.1. Descripción del problema

Primero usaremos un conjunto de datos para clasificación, consistente en 58509 instancias de 49 atributos obtenidos de señales eléctricas de motores.[5]

Nota. La descripción del *dataset* que empleamos no es demasiado exhaustiva. Las medidas son el resultado de una descomposición en modelo empírico (EMD) de las mediciones de corriente sin procesar. Además, no disponemos de información sobre tamaños de *train* ni de *test*.

Al igual que en el caso del problema de regresión, los datos están dispuestos en filas, correspondiendo las primeras 48 columnas a atributos de las señales eléctricas, y la última de ellas a la categoría correspondiente. Estamos en este caso ante un problema de **clasificación multiclase** en el que $\mathcal{X} = \mathbb{R}^{48}$, $\mathcal{Y} = \{1, ..., 11\}$ y queremos aprender $f : \mathcal{X} \longrightarrow \mathcal{Y}$ (desconocida) que, a cada conjunto de atributos del motor, le asigna su categoría correspondiente.

Profundizaremos en nuestros datos mediante su visualización. Veremos varias cosas:

- Ver si las distribuciones de clases son homogéneas. Para ello, haremos una gráfica de barras de los datos.
- Visualizaremos los datos mediante una técnica de reducción de dimensionalidad, que nos ayude a ver si los datos están agrupados.

En primer lugar, vemos mediante la gráfica de barras que las clases no están desbalanceadas.

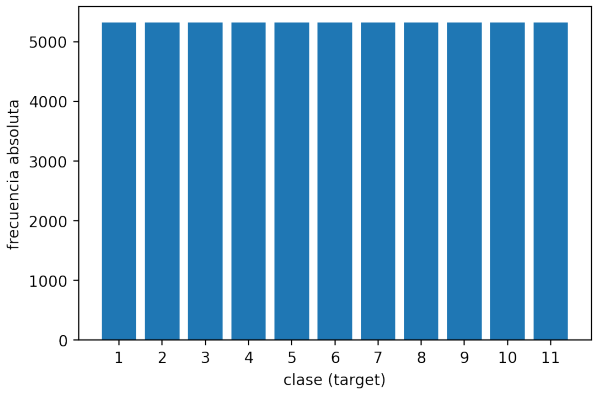

Figura 7: Distribución de clases en los datos de clasificación

A continuación, veamos cómo estan agrupados nuestros datos. Para ello usaremos la técnica t-SNE (*t-distributed stochastic neighbor embedding*), que proyecta nuestros datos a dos dimensiones.

[5] AutASS, 2011. Autonomous Drive Technology by Sensor Fusion for Intelligent, Simulation-based Production Facility Monitoring and Control: BMWi-funded Research Project, Grant Number: 01MA09006A.

Para ello, en primer lugar crea una distribución de probabilidad sobre parejas de muestras en el espacio de datos, para que las parejas similares (en cuanto a distancia y densidad en las proximidades de un punto) tengan alta probabilidad de ser escogidas. A continuación, lleva los puntos del espacio original a un espacio de baja dimensionalidad, mediante la minimización de la divergencia de Kullback-Leibler entre dos distribuciones respecto a los posiciones de los puntos en el mapa (esta divergencia mide la similitud o diferencia de las distribuciones de probabilidad).

En `scikit-learn` disponemos de una implementación de este algoritmo. Tras ajustar algunos parámetros, obtenemos la siguiente representación:

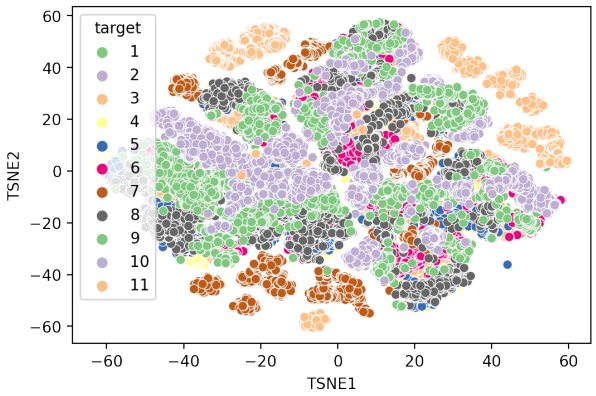

Figura 8: Diagrama t-SNE (en 2D) para problema de clasificación

Independientemente de los parámetros usados para t-SNE, no me ha sido posible encontrar una representación en el que las clases se vean más agrupadas.

3.2. Selección de $\mathcal{H}$

Para este caso seguiremos un razonamiento similar al problema de regresión y usaremos una clase de funciones de combinaciones cuadráticas de las observaciones. Dado que en el preprocesado haremos reducción de dimensionalidad, lograremos no aumentar demasiado la dimensión de cada punto. Además, el modelo será mucho más flexible para ajustarse a los datos, que intuimos que no son linealmente separables.

Siguiendo un razonamiento similar al caso de regresión, trabajaremos en un espacio transformado $\mathcal{Z}$, de dimensión

$$\tilde{d} = 1 + 2d' + \frac{d'(d'-1)}{2}, \text{ con } d' \leq d = 48$$

Escojamos a continuación la clase de funciones. Estamos en un problema multiclase de clasificación, para lo cual usaremos la técnica *one-versus-all*. Consideremos para cada $i \in \mathcal{Y} = \{1, ..., 11\}$ su clasificador binario en $\mathcal{Z}$:

$$h_i(z) = w^T z, \quad w \in \mathbb{R}^{\tilde{d}}$$

Este clasificador, salvo normalización con $\frac{1}{\|w\|}$, mide la distancia con signo de z al hiperplano que define w. Si la distancia es positiva, etiquetaremos a favor de la clase i. Si queremos predecir la clase de una instancia $z \in \mathcal{Z}$, la asignaremos a la clase que tenga una clasificación positiva

más fuerte o la que esté más cerca de la frontera de clasificación, es decir, le asignaremos la clase

$$\arg\max_{i \in \mathcal{Y}} h_i(z)$$

Obteniendo la clase de funciones para nuestro problema:

$$\mathcal{H} = \left\{ \arg\max_i w_{(i)}^T \Phi_2(x) : w_{(i)} \in \mathbb{R}^{\tilde{d}}, i \in \mathcal{Y} \right\}$$

En definitiva, tendremos un modelo con un total de $|\mathcal{Y}| \cdot \tilde{d} = 11\tilde{d}$ parámetros (un clasificador binario por clase).

3.3. Selección de conjuntos de entrenamiento, validación y *test*

Haremos exactamente igual que en el caso del problema de regresión: usaremos `train_test_split` para separar el conjunto de datos en un 20 % de *test* y un 80 % de *train*, evitando sesgos. Haremos *K-fold cross validation*, por lo que no seleccionaremos conjuntos de validación.

3.4. Preprocesado

Para el preprocesado de este problema seguiremos dos estrategias: en una de ellas reduciremos la dimensionalidad con PCA y en otra con ANOVA. Veamos a continuación los pasos de preprocesado:

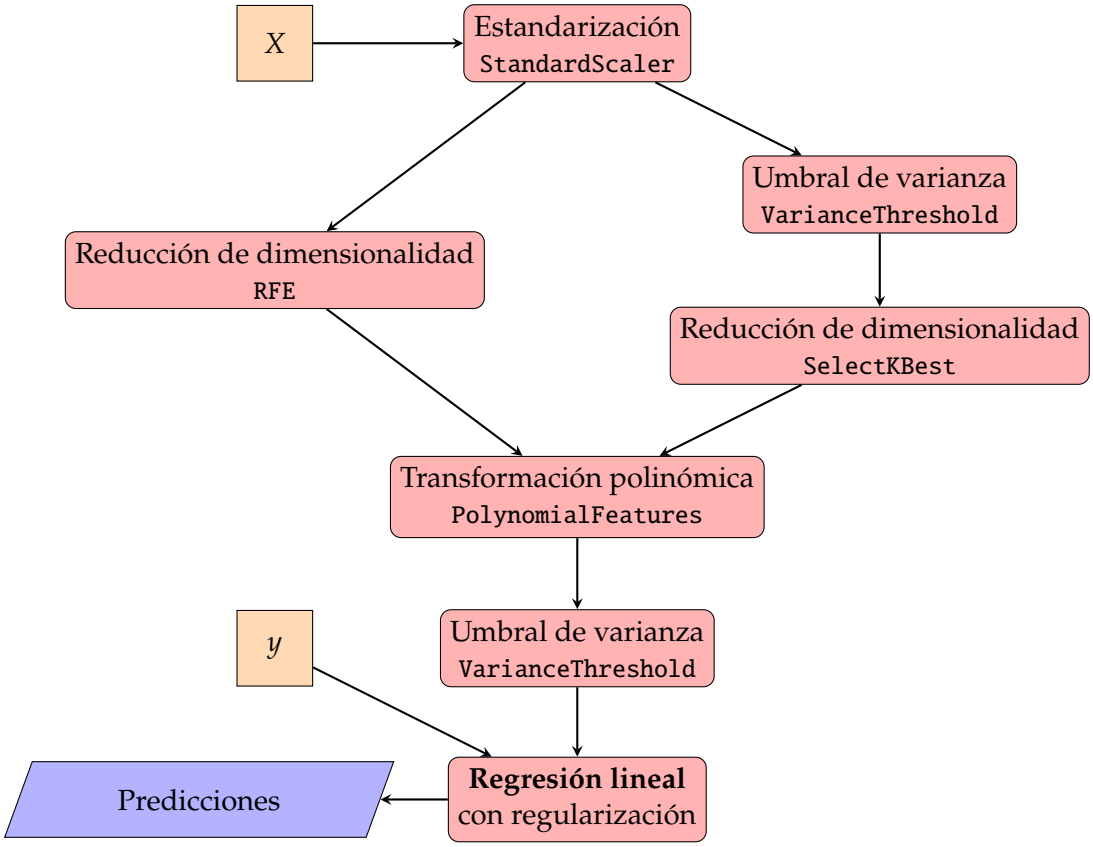

Figura 9: Diagrama de preprocesado para el problema de clasificación

Vemos dos *pipelines* de preprocesado diferentes, que se comentarán más adelante de forma más exhaustiva.

Una vez que aplicamos preprocesado, podemos ver cómo la matriz de correlación varía, pasando a tener variables menos correladas. Es interesante observar que los atributos están inicialmente muy correlados. A continuación figuran las matrices de correlación antes y después del preprocesado, para cada una de las estrategias.

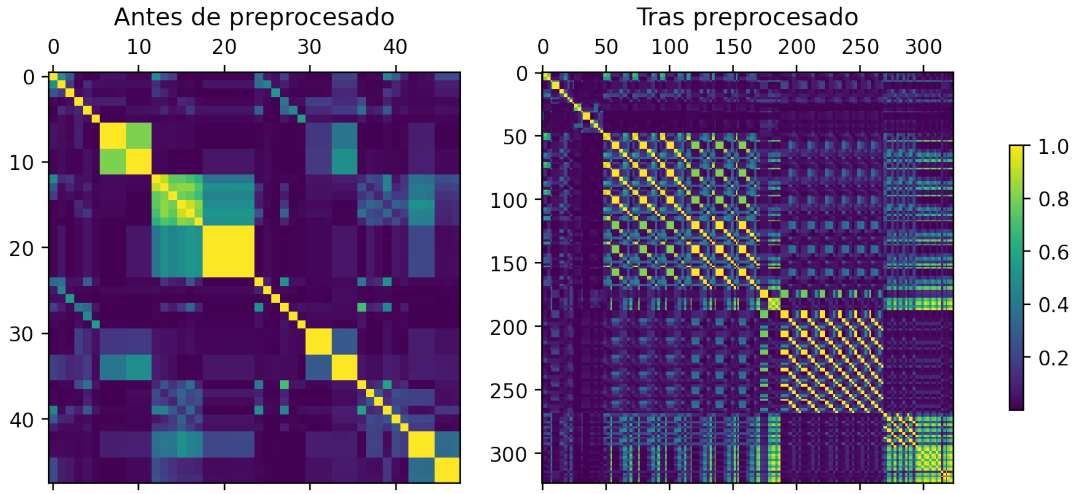

Figura 10: Matriz de correlación antes y tras el preprocesado, variante RFE

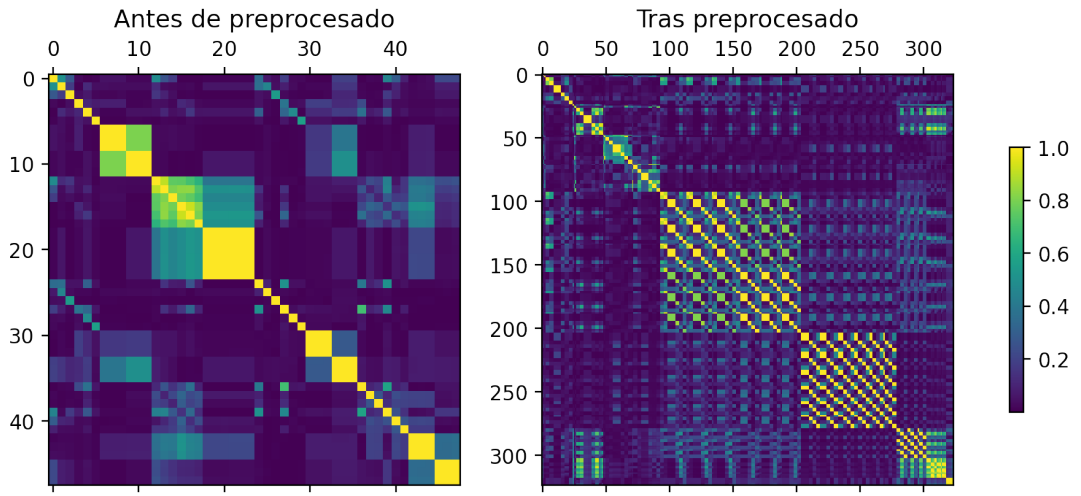

Figura 11: Matriz de correlación antes y tras el preprocesado, variante ANOVA

Dado que los diferentes pasos del preprocesado se comentaron exhaustivamente en el caso del problema de regresión, especificaremos a continuación las diferencias más relevantes de estos nuevos *pipelines*.

3.4.1. Estandarización

En este caso, también debemos de estandarizar para poder aplicar los algoritmos de reducción de dimensionalidad (al igual que ocurre con PCA y explicamos en el problema de regresión).

3.4.2. Reducción de dimensionalidad

En este caso aplicaremos dos algoritmos diferentes (se probó PCA con resultados mediocres). Por un lado, usamos RFE (tal y como se sugiere en un paper[6]) y por otro usaremos ANOVA.

El primero de ellos, RFE (*recursive feature elimination*), selecciona características considerando de forma recursiva sets de características cada vez más pequeños, hasta que llegamos a un cierto número de características – en nuestro caso 24, la mitad de ellas.

El segundo de ellos, ANOVA, se basa en el análisis de varianza. Para poder seleccionar las mejores variables utilizaremos *F-tests*, que usan el grado de dependencia lineal entre pares de variables, y ordenan las variables de más a menos discriminativas. Para poder aplicar ANOVA correctamente será necesario eliminar los atributos de varianza nula (y es por ello por lo que en este *pipeline* incorporamos adicionalmente un umbral de varianza (que, por defecto, es 0).

Mediante estas dos alternativas podremos estudiar cuál de estos preprocesados es más fino a la hora de ajustar nuestros datos.

3.4.3. Transformaciones polinómicas

Como dijimos anteriormente, hemos seleccionado una $\mathcal{H}$ con transformaciones polinomiales cuadráticas, por lo que usaremos `PolynomialFeatures(2)` en el preprocesado.

3.4.4. Umbral de varianza

De nuevo eliminamos las variables que tengan una varianza muy baja (inferior a 0.15).

3.5. Medida del error

Estando ante un problema de clasificación, la métrica que usaremos será el ***accuracy*** **(Acc)**, o grado de exactitud, que se calcula como sigue:

$$\text{Acc}(h) = \frac{1}{N}[\![h(x_n) = y_n]\!]$$

Esta métrica es adecuada porque prentendemos que nuestro clasificador sea preciso, y porque nuestras clases están balanceadas.

3.6. Regularización

Usaremos los mismos regularizadores que en el caso del problema de regresión, pero haciendo una elección concreta en el caso de uno de los algoritmos.

[6]Grüner, T., Böllhoff, F., Meisetschläger, R., Vydrenko, A., Bator, M., Dicks, A., ; Theissler, A. (2020, October 2). *Evaluation of Machine Learning for Sensorless Detection and Classification of Faults in Electromechanical Drive Systems*. Procedia Computer Science. https://www.sciencedirect.com/science/article/pii/S1877050920320706.

3.7. Selección de algoritmos

Como ya se ha explicado, usaremos *K-fold cross validation*. En el caso de nuestro problema, usaremos los siguientes modelos lineales. Todos ellos tratarán de maximizar `accuracy`.

- **`SGDClassifier con loss='hinge'`**: SVM con kernel lineal. Es un algoritmo que usa hiperplanos para separar el espacio, de modo que la separación entre el hiperplano y las observaciones sea máxima. En este caso usaremos la regularización `l2`, propia de SVM. Probaremos sin embargo con diversos valores de λ:
  - **Constante de regularización.** Probamos con 5 valores diferentes de λ, igualmente espaciados (logarítmicamente) en $[10^{-5}, 10^{5}]$.
- **`SGDCLasiffier con loss='lr'`**: regresión logística con gradiente descendiente estocástico. Este algoritmo ya fue explorado con detalle en prácticas anteriores. Usaremos las combinaciones de los siguientes parámetros:
  - **Regularización.** Usaremos tanto regularización `l2` como `l1`.
  - **Constante de regularización**. Probamos con 5 valores diferentes de λ, igualmente espaciados (logarítmicamente) en $[10^{-5}, 10^{5}]$.

3.8. Análisis de resultados

Vamos a comentar los resultados a continuación, teniendo en cuenta que diseñamos dos estrategias de preprocesado de los datos. Mediante RFE, el mejor modelo ha sido regresión logística con regularización L_1 y $\lambda = 10^{-5}$. Mediante ANOVA, el mejor modelo ha sido SVM con constante de regularización $\lambda = 3.162278 \cdot 10^{-3}$. Veamos en ambos casos que la regularización es bastante suave. Por otra parte, podemos ver los siguientes resultados en *accuracy*:

	Acc_{cv}	Acc_{in}	Acc_{test}	Tiempo
RFE	88.008404 %	79.960262 %	80.20851137 %	1452.996498 s
ANOVA	85.555716 %	83.797295 %	83.686549 %	1033.802724 s

3.8.1. Selección de la mejor hipótesis

Dado que los mejores resultados los obtenemos con RFE y regresión logística con regularización L_1, fijaremos este modelo como nuestra g definitiva.

Veamos a continuación el desempeño de nuestra g. A continuación, podemos ver la matriz de confusión del clasificador g en los conjuntos de *train* y *test*. Podemos ver que los errores se concentran en unas ciertas clases, siendo las que más la 2 y la 10.

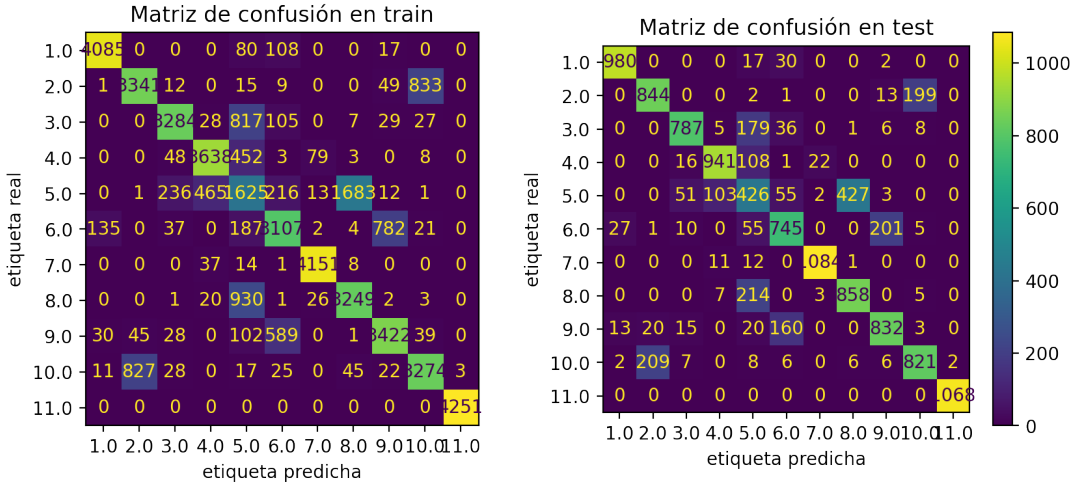

Figura 12: Matriz de confusión para los conjuntos de *train* y *test*

Por otra parte, vemos unas conclusiones similares para las gráficas que aparecen a continuación, especialmente para las dos útimas gráficas. En la primera gráfica vemos un pico en un número de datos concreto, que puede deberse a un factor aleatorio.

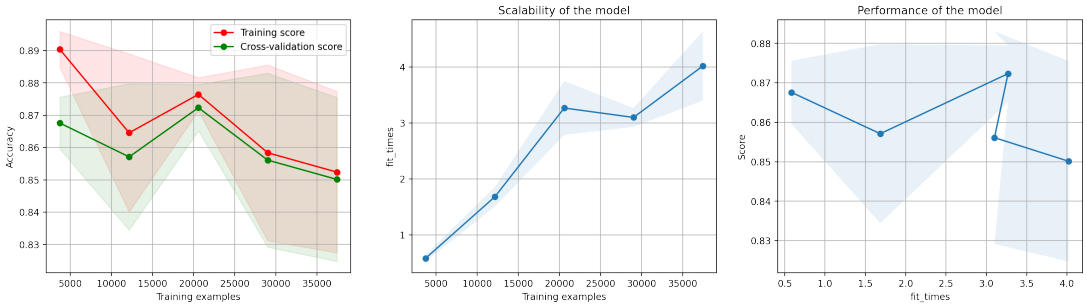

Figura 13: Curvas de aprendizaje para el problema de clasificación

3.8.2. Estimación de E_{out}

Finalmente, para estimar el error de nuestro modelo, recordamos cómo comentamos el hecho de que E_{cv} es buen estimador de E_{out}. Sin embargo, la mejor cota que podemos obtener en función de la desigualdad de Hoeffding la obtendremos a partir de E_{test}; tengamos en cuenta que no hemos usado el conjunto de *test* en ninguna fase de entrenamiento y selección. Dicha cota queda como sigue:

$$E_{out}(g) \leq E_{test}(g) + \sqrt{\frac{1}{2N_{test}} \log \frac{2}{\delta}} \text{ con probabilidad } \geq 1 - \delta$$

Tomando, por ejemplo, $\delta = 0.05$ podemos garantizar al 95 % de confianza que

$$E_{out}(g) \leq 0.1632 + \sqrt{\frac{1}{2 \cdot 11702} \log \frac{2}{0.05}} = 0.1715$$

En otras palabras, podemos decir que el modelo que hemos obtenido tiene un error del 17.15 % al 95 % de confianza.

3.9. Conclusiones

Para analizar la calidad del modelo obtenido haremos como en el caso de regresión, con ambas variaciones. Los resultados que obtenemos son los siguientes:

	Acc_{in}	Acc_{test}	Tiempo
Modelo conseguido	79.960262 %	80.20851137 %	1452.996498 s
Sin polinomios	62.116777 %	62.408135 %	73.12 s
Dummy	9.289209 %	9.246283 %	1.14 ms

Las conclusiones de nuestro modelo son similares al caso de regresión, y podemos ver cómo hemos conseguido un *accuracy* muy bueno, y claramente superior al medio y al obtenido si no usásemos transformaciones polinómicas.

Siguiendo un razonamiento análogo al anterior, podemos concluir que tenemos un modelo considerablemente bueno.