Práctica 3: Ajuste de modelos lineales

Alejandro Alonso Membrilla

Contents

Introducción

En esta práctica nos enfrentaremos a dos problemas de Aprendizaje Automático supervisado, uno de clasificación y otro de regresión, que intentaremos resolver de la mejor forma posible utilizando modelos lineales.

Nuestro estudio comprenderá desde la separación del dataset en conjuntos de entrenamiento y test, el análisis exploratorio de los datos y la transformación y limpieza de los mismos, hasta la selección de modelos apropiados y el análisis de los resultados obtenidos.

El código implementado para los experimentos ha sido escrito en Python, haciendo uso de la biblioteca `Numpy` para álgebra lineal y generación de números pseudo-aleatorios, `Scikit-Learn` para técnicas y herramientas de estadística y aprendizaje automático, y `Matplotlib` y `Seaborn` para la visualización de los resultados. Los cálculos y los gráficos generados para muchos experimentos en esta práctica han sido realizados en el ordenador personal del autor de la misma, Alejandro Alonso Membrilla, que cuenta con las siguientes características:

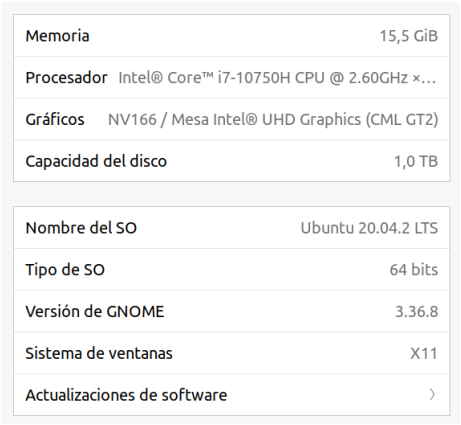


Figure 1: Especificaciones del hardware utilizado

En lo que respecta a la ejecución de algoritmos que hacen uso de una semilla, para garantizar la replicabilidad de nuestros resultados dicha semilla se ha fijado a un valor al principio del código implementado, se utiliza en todos los algoritmos pseudo-aleatorios y no se cambia en ningún momento a lo largo de la ejecución.

1 Problema de Clasificación

Descripción del problema

En este apartado trabajaremos con un conjunto de datos correspondientes a señales provenientes de sistemas de conducción electromecánicos, que puede descargarse libremente desde su repositorio en UCI. Cada elemento de nuestro dataset proviene de una serie de medidas con una sonda de corriente y un osciloscopio que han sido posteriormente descompuestas en distintas características mediante el *Empirical Mode Decomposition (EMD)*, de uso habitual en el análisis de series temporales. Los detalles del experimento y del EMD pueden leerse en este artículo y en las referencias que ahí se citan, aunque quedan fuera del alcance de esta práctica.

Nuestros datos están etiquetados con un número del 1 al 11, que indica el estado de las componentes del sistema que se ha estudiado. Desde el punto de vista del Aprendizaje Automático, este problema de detección de fallos resulta en uno de **clasificación supervisada sobre vectores reales**, esto es, en el que a partir de unos datos etiquetados queremos aproximar una función $f\colon \mathbb{R}^d \to \{1, ..., 11\}$, donde d es el número de características de nuestro conjunto de datos, que aplica cada vector de "medidas" en la etiqueta que le correspondería a un sistema del tipo de los estudiados donde se midiesen unas características como estas.

Clases de funciones

Dado que no tenemos información a priori sobre la distribución de los datos con los que vamos a trabajar (su interpretación física y matemática es desconocida para nosotros sin la ayuda de un experto), necesitamos experimentar con modelos suficientemente versátiles, o en su defecto varias clases de funciones con complejidades distintas que permitan encontrar aquella hipótesis que equilibre sesgo y varianza en su solución de este problema.

Como para esta práctica nos estamos limitando al uso de modelos lineales, escogeremos las clases de funciones dadas por **Regresión Logística** y **SVM** con un término de regularización y utilizando un kernel lineal. Ambos métodos permiten distintos tipos y escalas de regularización, lo que es deseable para nuestro experimento. Otros argumentos a favor de estas clases de funciones son:

1. Regresión Logística admite una extensión natural a problemas multiclase como este, denominada RL multinomial, que sustituye la función sigmoidal (que transforma el resultado de la regresión en una probabilidad de pertenencia a una clase) por la *softmax*, análoga a la anterior pero que da una probabilidad de pertenencia a cada clase por separado. Al contrario que otros algoritmos, que necesitan tratar los problemas con varias clases como varios problemas binarios, la variante multinomial permite generar una distribución conjunta mediante un método de máxima verosimilitud (detalles en las diapositivas de teoría). Este método parece ser preferido cuando el número de clases es fijo y todas son mutuamente excluyentes entre sí, como es nuestro caso.
2. SVM es un algoritmo ampliamente utilizado, aunque a menudo con kernels no lineales, debido a que gracias a su formulación matemática permite encontrar hiperplanos separadores que son óptimos en cierto sentido apropiado para la clasificación (maximiza la distancia de los vectores soporte al hiperplano). Sumado a sus opciones para la regularización y a cierto grado de tolerancia al ruido, esto convierte al SVM en una opción mucho más atractiva que el perceptrón o la regresión lineal.

Ahora debemos considerar la posibilidad de incluir **transformaciones no lineales** sobre nuestros datos. Aplicar transformaciones polinómicas sobre las características originales podría llegar a suponer, si el problema no resulta ser puramente lineal, una gran ventaja en tanto que estamos utilizando modelos lineales. Sin embargo, a priori, no podemos descartar el uso de las características originales, ni tampoco justificar que unas transformaciones sean mejor que otras.

Por tanto, supongamos el caso en que decidimos conservar las variables originales, e ir añadiendo transformaciones polinómicas sobre ellas de orden superior, es decir, primero las cuadráticas, luego las cúbicas... conservando siempre las anteriores. Buscamos una forma de añadir dimensionalidad a nuestro dataset, pero de una forma que siga un crecimiento progresivo de complejidad. Nos justificamos en que un modelo que explique bien los datos usando características lineales y cuadráticas es más sencillo, y por tanto más deseable, que uno que lo haga a partir de características de orden 30. Claramente, el cálculo del número de posibles combinaciones (con repetición) de k de las d variables nos indica que el número total de características (NTC) añadiendo transformaciones de hasta grado k es el siguiente:

$$NTC(d,k) = \sum_{i=1}^{k} \binom{d+i-1}{i}$$

Para elegir un $k \geq 1$, grado máximo de las transformaciones polinómicas, aplicaremos un criterio de simple capacidad computacional. No nos importa aumentar mucho la complejidad de nuestros modelos a través del aumento del número de dimensiones, puesto que no conocemos la complejidad del problema y ya estamos considerando el uso de regularización para simplificar nuestras hipótesis. Por tanto, simplemente sustituiremos el número de variables de nuestro conjunto de datos en la fórmula anterior, y tomaremos el primer valor de k que no haga computacionalmente imposible la resolución de la práctica en un tiempo apropiado usando el hardware disponible (ver Introducción).

Destacamos que esto **no se trata de *data snooping***. En primer lugar, no vamos a utilizar ninguna información de nuestro dataset ajena a su tamaño para fijar las transformaciones no lineales. En segundo lugar, ya hemos dejado claro que nuestro criterio no es escoger las transformaciones no lineales que obtengan un menor error para el número de variables que observemos (si es que eso es si quiera posible, conociendo solamente el número original de variables), sino que será únicamente elegir el orden máximo que sea computacionalmente

viable para nuestro experimento. Este criterio es totalmente independiente al enfoque teórico del Aprendizaje Automático, y por tanto no puede considerarse *data snooping*.

Nuestro conjunto de datos cuenta con 48 características (más la etiqueta) y 58509 instancias (antes de separar en entrenamiento y test). Sustituyendo $d = 48$ en la fórmula obtenemos:

$$NTC(48,1) = \binom{48+1-1}{1} = 48$$
$$NTC(48,2) = 48 + \binom{48+2-1}{2} = 48 + 1176 = 1224$$
$$NTC(48,3) = 1224 + \binom{48+3-1}{3} = 1224 + 19600 = 20824$$

Para $k = 3$ ya observamos una explosión en el número de variables, que probablemente aumente nuestros tiempos de ejecución de forma sustancial. Tomando $k = 2$ el cálculo podría ser viable, siempre que pudiésemos reducir suficientemente los datos en fase de preprocesamiento. Partiremos de $k = 2$, pero nos reservamos la posibilidad de quedarnos con las variables originales si el preprocesamiento no es suficiente para reducir la dimensionalidad a unas cotas viables para el entrenamiento de los modelos.

Una opción a tener en cuenta hubiese sido aumentar mucho la dimensionalidad de nuestro dataset y restringir nuestros modelos a **formulaciones duales**. Esta formulación queda **descartada** al contar con más de 40000 ejemplos de entrenamiento, puesto que el coste computacional de resolver la formulación dual crece con el número de ejemplos.

Análisis Exploratorio y Preprocesamiento

En esta sección describiremos todo el proceso realizado con los datos previo al entrenamiento y selección de modelos. Nuestro objetivo final es poder evaluar de alguna manera el modelo final seleccionado. Para esto necesitamos separar un conjunto de test, $\mathcal{D}_{test}$, que no tomará parte alguna en el preprocesado ni en el entrenamiento de los modelos, con el que calcularemos una aproximación a nuestro error fuera de la muestra. El resto del dataset se denominará conjunto de entrenamiento, $\mathcal{D}_{train}$, y será el único que analizaremos en esta sección.

Cabe destacar que algunas de las transformaciones hechas sobre el conjunto de entrenamiento tendrán que replicarse sobre el conjunto de test para que tenga sentido aplicar sobre él los modelos entrenados. Siempre que esto suceda se indicará explícitamente.

Lectura y Separación de los datos

Nuestros datos son leídos de forma conjunta desde el fichero `Sensorless_drive_diagnosis.txt`. Comprobamos que no hay valores perdidos, y separamos los datos y sus etiquetas en estructuras de datos separadas.

Por último, separamos los datos de test mediante la orden `train_test_split()` de Scikit-Learn. El porcentaje de datos separados será del 20%, siguiendo las indicaciones dadas aquí. Un rápido vistazo a nuestros datos no indica que estos no se encuentran barajados, las instancias de cada clase se disponen secuencialmente, sin entremezclarse. Para corregir esto podemos llamar a la función anterior con el parámetro `shuffle` activado, y la semilla para fijar el resultado.

Para evitar una distribución desbalanceada de las clases entre entrenamiento y test, realizamos esta separación de forma estratificada. Con el objetivo de asegurarnos de haber hecho la separación de esta forma, la figura 2 muestra un gráfico de barras con el número de instancias por clase en $\mathcal{D}_{train}$ y $\mathcal{D}_{test}$. Dicho gráfico demuestra dos cosas. La primera es que, efectivamente, la proporción de cada clase en $\mathcal{D}_{train}$ y $\mathcal{D}_{test}$ se mantiene para todas las clases. La segunda es que todas clases cuentan con, aproximadamente, el mismo número de instancias, por lo que el problema está perfectamente balanceado.

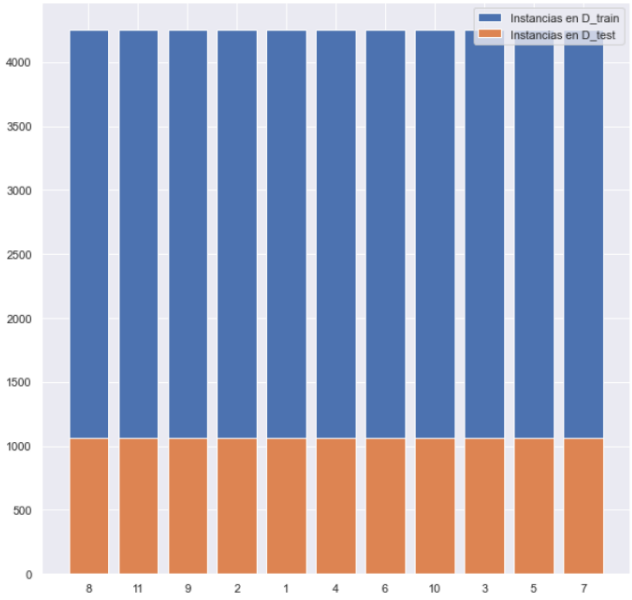


Figure 2: Número de instancias por clase en $\mathcal{D}_{train}$ y $\mathcal{D}_{test}$.

Análisis descriptivo y selección de características

Muchas técnicas del Aprendizaje Automático asumen o funcionan mejor cuando los datos se encuentran normalmente distribuidos. Por ejemplo, el F-Test de ANOVA permite encontrar dependencias entre distintas variables, pero asume una distribución normal de los datos. Algoritmos como el Análisis Lineal Discriminante son computacionalmente eficientes y permiten cierta interpretabilidad de sus resultados, pero también asumen una distribución gaussiana de las variables independientes como hipótesis fundamental.

Para plantearnos usar o descartar técnicas como estas, empezaremos aplicando un test de normalidad sobre $\mathcal{D}_{train}$. Para ello, consideramos como hipótesis nula que cada una de nuestras variables vengan de una distribución normal univariante. El test utilizado es el proporcionado por SciPy y los resultados son claros. Para todas las variables, la hipótesis nula se rechaza con un p-valor menor a $2 \cdot 10^{-191}$. Es decir, con casi toda seguridad **nuestras variables no siguen una distribución normal**.

Tras analizar nuestras características originales, aplicamos las **transformaciones cuadráticas** establecidas en la sección anterior. Es importante aplicar las mismas transformaciones sobre el conjunto de test, puesto que el modelo lineal que entrenaremos ahora asumirá un dato del espacio de características transformado.

A continuación realizaremos un análisis de la colinealidad entre las diferentes características. Muchos algoritmos de aprendizaje asumen independencia o incorrelación entre las variables, o al menos funcionan mejor en estas circunstancias. Eliminar las variables correladas puede mejorar los resultados de nuestros modelos al mismo tiempo que eliminamos información redundante, que solamente lograría retrasar nuestro aprendizaje.

En los datos podemos encontrar dos tipos de dependencias: lineales y no lineales. En nuestro caso, hemos añadido a propósito ciertas variables con dependencias no lineales que no nos gustaría eliminar. Además, dado que usaremos únicamente modelos lineales, difícilmente estos encontrarán alguna redundancia entre características que solo guarden una relación no lineal. Por esto, descartamos métodos como el de Spearman para calcular dependencias generales y elegimos el método de Pearson, que busca correlaciones lineales en los datos. Cabe destacar que dicho método es invariante a escala y traslaciones.

Nuestro método para seleccionar características consistirá en calcular los **coeficientes de correlación de Pearson** para cada dos variables y eliminar aquellas características para las que ya exista otra dentro del dataset con la que correlacionen fuertemente. En este artículo se explica que una correlación en valor absoluto mayor o igual a 0.7 se considera "fuerte". En nuestro caso, eliminaremos toda correlación mayor a 0.9 para garantizar que no perdemos información importante. La figura 3 compara las dependencias lineales de nuestro conjunto de entrenamiento antes y después de eliminar variables redundantes. Vemos que el número de variables ha decrecido significativamente, de 1200 a menos de 150.

En el procedimiento anterior hemos eliminado características que suponían información redundante dentro

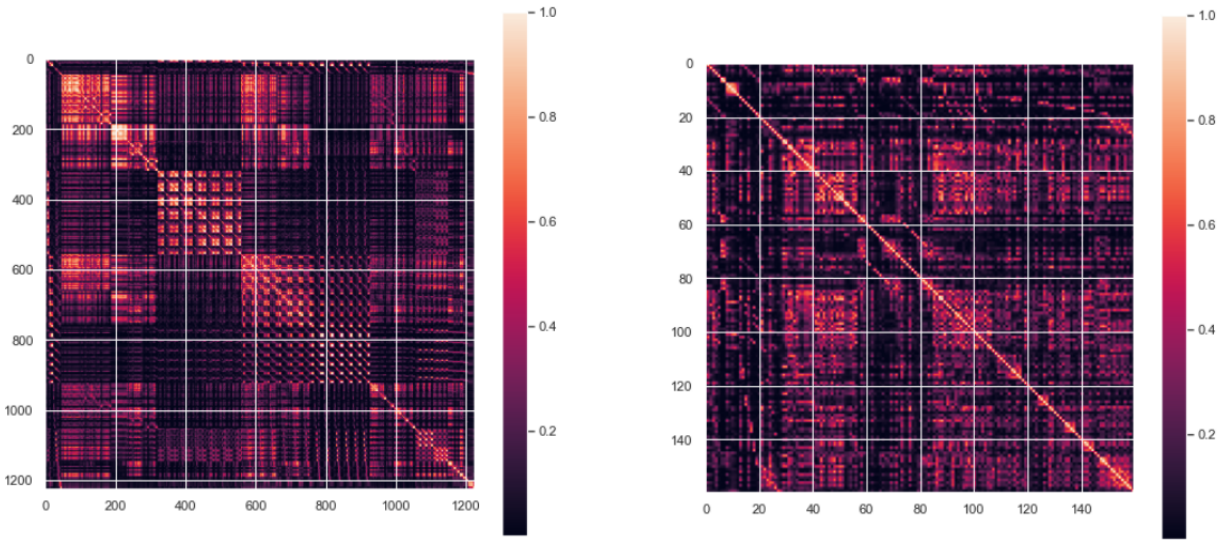


Figure 3: Correlaciones lineales entre características antes (izquierda) y después (derecha) de eliminar variables redundantes.

de la muestra, sin tener en cuenta las etiquetas de cada instancia. Creemos que podría ser interesante eliminar también variables que no supongan de importancia a la hora de resolver el problema de clasificación que nos estamos planteando, o que al menor no parezcan útiles.

Nos gustaría tener una forma de medir cuanta información parece aportar cada variable para resolver nuestro problema. Al contar con un dataset con un ratio instancias/atributos relativamente alto, es posible aplicar métodos no paramétricos para estimar la dependencia entre las etiquetas y cada uno de los atributos. Esto es lo próximo que vamos a probar, con la función `mutual_info_classif()` de Scikit-Learn. Para variables continuas, esta función agrupa los distintos valores observados usando un esquema de K-NN, discretizando el espacio, y calculando **una medida de información mutua** para variables discretas tal como se puede ver en este artículo. Al considerar la información mutua de cada variable por separado, el método puede ser invariante a centro y escala y no es necesario normalizar los datos previamente.

El resultado de aplicar `mutual_info_classif()` sobre cada variable es un número real entre el 0 y el 1, que es más alto cuanta mayor dependencia haya entre dicha variable y el vector de etiquetas. El número de vecinos utilizado para el agrupamiento mediante el vecino más cercano ha sido fijado en 3 vecinos, justificándonos en los resultados expuestos en teoría que abalan este valor empíricamente (tema 9, pág. 16). La gráfica en la figura 4 representa los resultados.

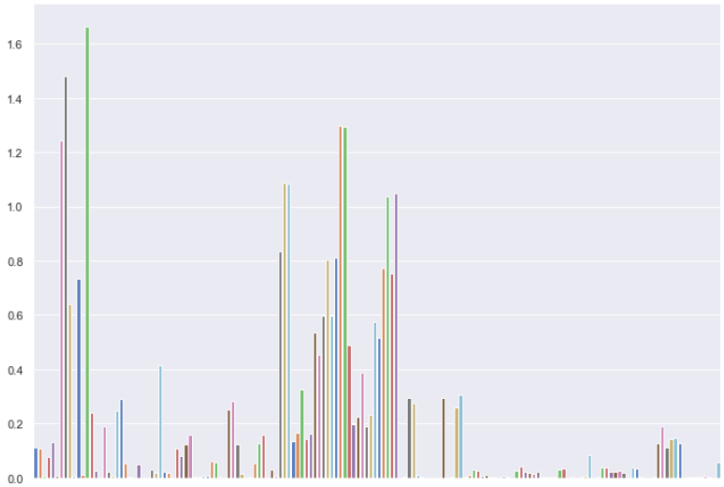


Figure 4: Información mutua de cada variable con el vector de etiquetas.

Observamos un gran número de variables con una información mutua muy baja y que podría ser conveniente

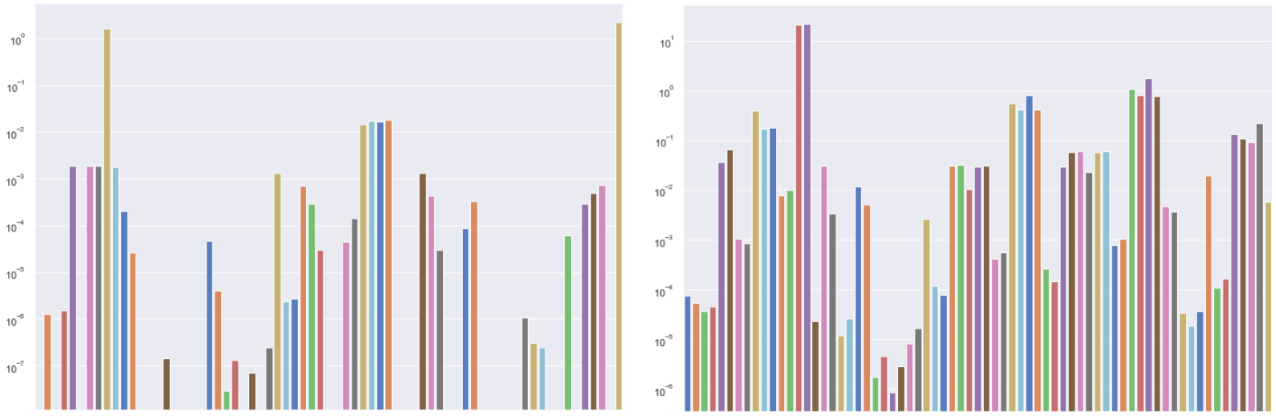
eliminar. Sin embargo, es necesario tener en cuenta que este método solo considera dependencias **univariables** con el vector de etiquetas, por lo que variables con poca información mutua podrían ser, en conjunto, más importantes de lo que cabría esperar tras las medidas anteriores. En pos de evitar eliminar injustamente más características de lo que sería conveniente, vamos a establecer un umbral relativamente bajo de 0.05 (un 3% de la información mutua máxima observada), que usaremos para filtrar variables poco importantes.

La selección de características realizada, tras eliminar variables redundantes y con baja información mutua, nos deja con un total de 69 variables. Es importante seleccionar en el conjunto de test las mismas variables que han resultado de este procedimiento, puesto que son las que nuestros modelos aprenderán a interpretar.

Normalización

Diferencias notables en escala entre variables pueden suponer un problema para muchos métodos en Aprendizaje Automático. En este artículo se indica que algoritmos que "ajustan un modelo a partir de sumas ponderadas de los *inputs*", como Regresión Logística, o que "usan medidas de distancias", como SVMs, se ven afectados por la normalización. Esto es especialmente cierto cuando se desea usar algún tipo de regularización sobre los pesos, puesto que escalas muy distintas en los datos implican diferencias artificiales en los pesos que dificultan la optimización con restricciones sobre dichos pesos y provocan inestabilidad (*The Elements of Statistical Learning* cap. 3.4.1).

Vamos a analizar si es necesario normalizar nuestro conjunto de datos. La figura 5 nos muestra las medias y desviaciones típicas de los datos en $\mathcal{D}_{train}$, con las características restantes después de la selección del apartado anterior. Nótese que la escala es logarítmica.



Medias Desviaciones típicas

Figure 5: Medias y desviaciones típicas en escala logarítmica de nuestro conjunto de entrenamiento

La figura 5 deja patente la diferencia en escala entre las distintas variables. Normalizar nuestro conjunto de datos es obligatorio. Además, a la hora de visualizar nuestras variables en busca de *outliers*, normalizar será sumamente conveniente.

Existen dos opciones ampliamente usadas para normalización en Machine Learning. La primera es restarle a cada variables el mínimo de los valores que toma, y dividir entre la longitud de su rango, es decir, entre la diferencia entre su máximo y su mínimo. La segunda opción es restarle la media de los valores que toma y dividirla entre su desviación típica. En este caso elegiremos la segunda, conocida también como estandarización. En este y este artículo se expone que este método es más robusto que el primero ante valores extremos. En este otro se defiende que estandarizar tiende a obtener mejores resultados en las SVMs. Apoyándonos en estas referencias, decidimos este como nuestro método de normalización.

La transformación aplicada sobre el conjunto de entrenamiento debe replicarse sobre el de test. Esto no quiere decir restarle al conjunto de test su media y dividir entre su desviación típica, sino restarle la media del conjunto de entrenamiento y dividirlo entre la desviación típica de este mismo conjunto. En resumen, es necesario aplicar al conjunto de test **exactamente las mismas transformaciones** que aplicamos sobre el de entrenamiento.

La figura 6 muestra un diagrama de cajas con las variables estandarizadas. Un rápido vistazo a la escala

muestra una distribución algo extraña de los puntos en la gráfica. La mayor parte de las características toman valores entre -20 y 20, pero la escala se extiende hasta ±200 debido a valores que se alejan muy notablemente del resto. Estos será, muy probablemente, valores extremos o *outliers* que trataremos a continuación.

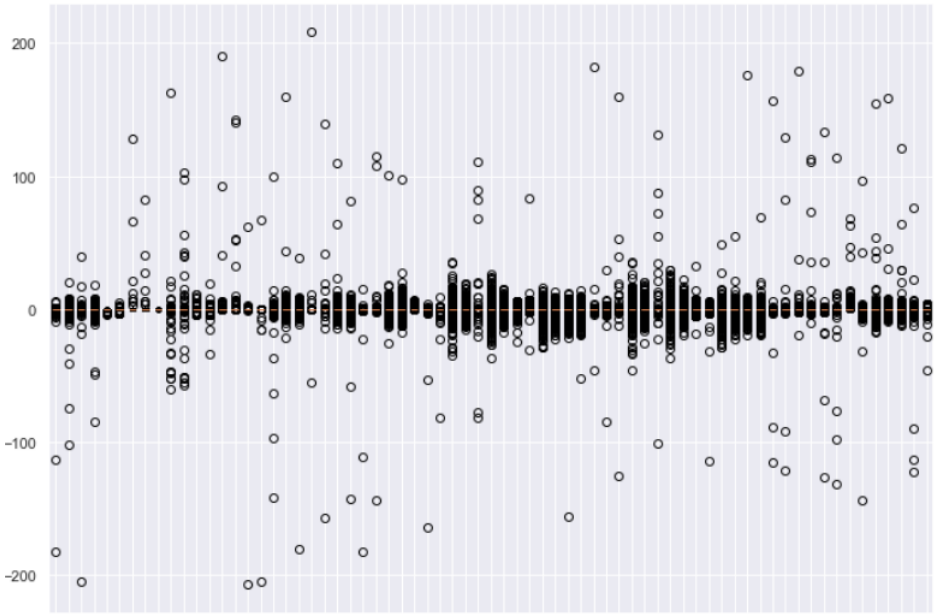


Figure 6: Diagrama de cajas de las variables de entrenamiento tras normalizar.

Tratamiento de valores extremos

No conocemos en profundidad la naturaleza del origen de nuestros datos. Esto hace que no podamos diagnosticar eficazmente las causas de los valores extremos que observamos en la figura 6. Sí podemos, sin embargo, tomar medidas heurísticas que tengan sentido desde el punto de vista del entrenamiento de los modelos, como eliminar valores "muy alejados" del resto de la distribución.

El criterio a seguir será el siguiente: fijamos un intervalo centrado en 0 para cada variable. Dicho intervalo tendrá una longitud de k veces el rango intercuartílico de las observaciones en dicha variable. Para cada valor de k desde 0 hasta 49 y contaremos cuantos puntos se encuentran dentro de este rango en todas las variables. Los resultados se muestran en la figura 7.

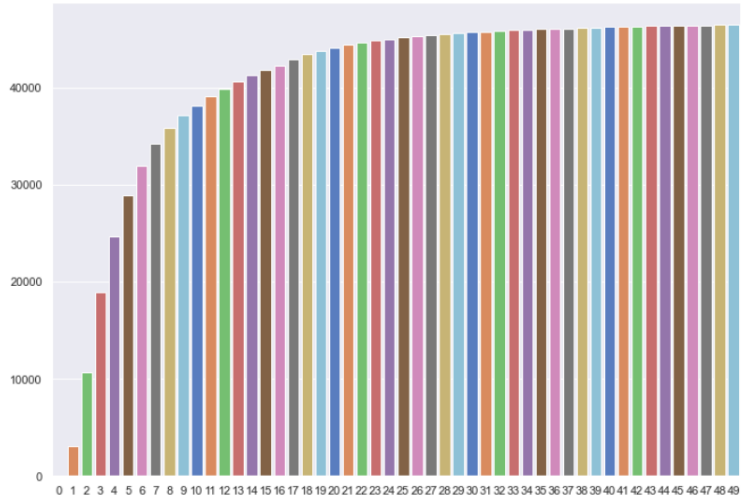


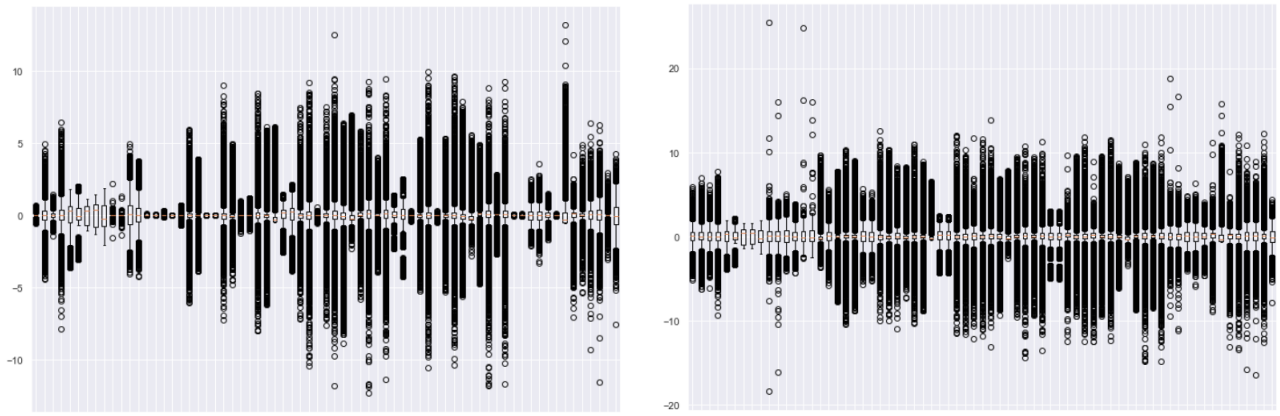
Figure 7: Número de valores no extremos para cada k.

Nuestro criterio será similar al *método del codo*, que suele emplearse junto al K-medias para seleccionar un valor de K apropiado. Este método consiste en, dada la gráfica en la figura 7, elegir un valor de k que, siendo

lo más pequeño posible y dejando los valores más extremos fuera, aúne una gran cantidad de puntos y cada k mayor a este añada muy pocas instancias nuevas. Esto suele visualizarse como un "codo" en la gráfica (de ahí el nombre).

Argumentamos que, aunque la elección del k adecuado sigue siendo cualitativa, este criterio lleva implícito cierto concepto de "normalidad", que lo hace apropiado para descartar *outliers* o valores "no normales". Si, a partir de cierto k, los puntos empiezan a ser mucho más dispersos, solitarios y lejanos, estos pueden considerarse como valores atípicos. El método del codo nos permite visualizar en cierto sentido la relación entre la dispersión de la distribución y la cantidad de puntos dispersos.

A partir de la explicación anterior y de la figura 7, tomamos $k = 25$. El número de puntos restantes es de 45109 puntos, es decir, hemos eliminado en torno al 3.63% de las instancias. El resultado tras eliminar *outliers* puede verse en la figura 8 (izquierda).



Sin renormalizar Renormalizados

Figure 8: Diagrama de cajas de nuestros datos separados por variables tras eliminar valores extremos.

Sin embargo, los valores extremos han afectado al proceso de normalización. Unos *outliers* muy destacados pueden hacer que el resto de valores de la variable se agrupen excesivamente para compensar la desviación producida por estos valores extremos. Además, al eliminarlos nuestro dataset deja de estar normalizado. Por esto es recomendable volver a normalizar (por supuesto, replicando la transformación sobre el conjunto de test). Los resultados pueden observarse en la figura 8. Notamos una escala más parecida entre las distintas variables, y que varias características que se encontraban muy concentradas en torno al 0 antes de renormalizar se han dispersado, como se esperaba.

Selección del Modelo final

El modelo final se elegirá evaluando cada opción mediante validación cruzada. Dados los datos de entrenamiento, cada modelo se entrenará 5 veces con cada posible combinación de parámetros (k-fold cross-validation con $k = 5$), cada vez eligiendo un subconjunto con una quinta parte de los datos que hará de conjunto de validación, para testear el modelo resultante de entrenar con los parámetros establecidos y el resto de datos de entrenamiento. Durante el proceso de validación cruzada, cada elemento de $\mathcal{D}_{train}$ es usado para validar exactamente una vez. Esto hace que cada dato tenga una única predicción asociada. Estas predicciones pueden compararse con las etiquetas reales para calcular un porcentaje de error de clasificación, que será nuestra principal métrica de error en el problema de clasificación.

K-fold y Métrica del Error

La **elección del valor de** k se ha realizado teniendo lo siguiente en consideración. Un número de *folds* muy alto reduce el sesgo a la hora de estimar E_{out} con respecto a uno bajo. El criterio de *Leave One Out* (Deja Uno Fuera) dará aproximadamente una estimación insesgada del error de test, como se ha visto en teoría. Sin embargo, este método posee una varianza muy elevada y, principalmente, es computacionalmente inviable para

conjuntos de datos grandes. Elegir un número de *folds* más reducido busca un ajuste entre el sesgo que esto introduce, la varianza de la estimación y el ahorro computacional.

De acuerdo con An Introduction to Statistical Learning, cap. 5.1.4, los valores de k más utilizados son $k = 5$ y $k = 10$, puesto que "se ha demostrado empíricamente que producen estimaciones del error de test que no sufren excesivamente de un sesgo ni de una varianza demasiado altos". Nos fundamentamos en esto para tomar $k = 5$, como se ha dicho anteriormente.

La **métrica del error fundamental** para guiar el proceso de selección es la del **porcentaje de ejemplos mal clasificados**, puesto que se considera que es la métrica más natural para un problema de clasificación. Además, Regresión Logística y SVM minimizan funciones de pérdida distintas, por lo que no tendría sentido comparar estas directamente.

Clasificación cuenta con **otras métricas de evaluación**, pero difícilmente estas pueden transformarse en un número para poder comparar resultados de forma sistemática, especialmente en el caso multiclase. Algunas son la matriz de confusión (una matriz), la precisión, el *recall* o la *F1-Score* (las tres son vectores, con una componente por clase). Una opción para estas tres últimas medidas es calcular la media de sus componentes, pero no vemos motivos para considerar que, en este caso, la media de estos valores pueda ser más representativa de la solución que la tasa de aciertos.

Tras esta discusión, el porcentaje de ejemplos mal clasificados parece la opción más apropiada. Esto no impide, sin embargo, que se analicen otras métricas del error, dada la solución final. En nuestro experimento de selección de modelos, seleccionaremos un modelo de Regresión Logística y otro de SVM lineal a partir de su número de ejemplos bien clasificados, y compararemos sus matrices de confusión.

Regularización

Nuestra elección de la regularización se fundamentará en el análisis exploratorio de los datos y en el preprocesamiento realizado en secciones anteriores. Consideraremos dos tipos de regularización, L1 (o LASSO) y L2 (o Ridge). Ambas consisten en añadir a la función de pérdida que se desea minimizar $L(w, p; x)$ (w representa los pesos, p un parámetro o conjunto de parámetros que se optimizan junto a los pesos, como podría ser el sesgo), relacionada de una forma u otra con las soluciones del problema de clasificación, un término dependiente del tamaño de los pesos. La función de pérdida resultante es

$$\lambda \frac{1}{2} ||w||_1 + L(w, p; x) \qquad (L1)$$

$$\lambda \frac{1}{2} w^T w + L(w, p; x) \qquad (L2)$$

Aquí, λ es un parámetro que modula la fuerza de la regularización. En Scikit-Learn, para RL y SVM se opta por una formulación equivalente:

$$\frac{1}{2} ||w||_1 + C \cdot L(w, p; x) \qquad (L1)$$

$$\frac{1}{2} w^T w + C \cdot L(w, p; x) \qquad (L2)$$

C es inversamente proporcional a la fuerza de la regularización. Cuanto menor sea, más peso tendrá el término regularizador sobre la expresión completa.

Destacamos que L1 tiende a eliminar variables de poca importancia, mientras que L2 se inclina a "mitigar el problema de la multicolinealidad", o correlaciones lineales multiples. En el preprocesamiento hemos aplicado selección de características mediante eliminación de correlaciones y de variables que tenían una baja información mutua con el vector de etiquetas. Destacamos el hecho de que el umbral para eliminar variables con información mutua baja se ha fijado muy bajo para eliminar características probablemente inútiles con bajo riesgo. Sin embargo, muchas de las variables no eliminadas contaban con una información mutua relativamente baja que podrían ser descartadas igualmente. Una regularización tipo LASSO podría aplicar una selección de características todavía más precisa.

Elegimos L1 como nuestra técnica de regularización, pero aún falta elegir posibles valores para el parámetro C. Antes de empezar el experimento, no es posible saber qué grado de regularización es el más conveniente puesto

que no conocemos la capacidad de RL ni de SVM lineal para ajustarse a nuestro conjunto de entrenamiento. Probaremos con un rango de valores de C que le dé más peso a la regularización (menor que 1), el mismo y menos peso a la regularización (mayor que 1). Tomaremos $C \in \{0.1, 0.5, 1, 5, 10\}$.

Otros Parámetros del Entrenamiento

Antes de comenzar nuestro experimento necesitamos elegir un *solver*, o algoritmo de ajuste para cada modelo.

1. Para **Regesión Logística**, Scikit-Learn nos ofrece un amplia gama de opciones: `newton-cg`, `lbfgs`, `liblinear`, `sag`, `saga` y `SGD`. Sin embargo, el único que permite regularización L1 y la función de pérdida multinomial es `saga`. Este es un método para optimizar sumas de funciones convexas de una forma eficiente, descrito en este artículo, basado en incremento de gradiente. Junto a `liblinear`, es el único en admitir regularización L1 gracias a funcionar incluso añadiendo términos convexos no diferenciables, como es la norma-1 (demostración en el artículo anterior).
2. En el caso de **SVM lineal**, para utilizar regularización L1 tenemos las opciones de `liblinear`, que usa un método de tipo *coordinate descent*, y `SGD`. `SGD` es un optimizador de propósito general, que no da garantías de optimalidad ni convergencia y requiere de un mayor esfuerzo a la hora de ajustar parámetros. Si bien pudiera ser la opción más conveniente con un dataset más grande en el que `liblinear` requiera de demasiado tiempo para converger, ese no es el caso. En conclusión, utilizaremos `liblinear` por permitirnos usar regularización L1, ser relativamente sencillo de configurar y aportar un análisis teórico sólido sobre su convergencia.

A raíz de la discusión anterior, decidimos usar las clases `LogisticRegression` y `LinearSVC` de Scikit-Learn para implementar nuestros modelos. Procedemos a explicar los parámetros de ambas clases:

Regresión Logística:

- `penalty`: especifica el tipo de regularización a usar. Lo fijamos a 'l1' para usar regularización de tipo L1.
- `dual`: indica si se usa la formulación dual o primal del problema de optimización. Puesto que tenemos más ejemplos que características, usaremos la formulación primal, es decir, dejamos `dual` a `False`.
- `tol`: tolerancia considerada como criterio de parada. Dejamos el valor por defecto, 0.0001, ya que disminuirlo podría hacer aumentar notablemente el tiempo de cómputo.
- `C`: como ya hemos visto, determina la intensidad de la regularización de forma inversamente proporcional. Se tomará $C \in \{0.1, 0.5, 1, 5, 10\}$.
- `fit_intercept`: añade un término de sesgo. Lo activamos, puesto que no se lo hemos incluido manualmente.
- `intercept_scaling`: fija el valor del sesgo. Lo dejamos a 1.
- `class_weight`: asigna un peso a cada clase. Sirve como mecanismo contra el desbalanceo de clases. Como no es nuestro caso, lo dejamos al su valor por defecto (todas las clases con peso 1).
- `randomState`: semilla para los procesos pseudo-aleatorios. Le pasamos nuestra semilla prefijada.
- `solver`: es el algoritmo usado para resolver el problema de optimización. Fijado a `saga`.
- `max_iter`: máximo número de iteraciones del 'solver'. Consideramos 10000 iteraciones para que tenga suficientes para converger pero no necesite demasiado tiempo de cómputo.
- `multi_class`: método para resolver problemas multiclase. Por los motivos justificados al fijar nuestra clase de funciones, fijaremos este parámetro a 'multinomial'.
- `warm_start`: toma como pesos iniciales los resultados de la ejecución anterior. Como queremos aplicar validación cruzada, esto es absurdo, puesto que estaríamos usando como pesos iniciales aquellos pesos aprendidos a partir de datos que ahora nos gustaría predecir (data snooping). Lo dejamos a `False`.
- `n_jobs`: número de hilos en paralelo en los que se realiza el entrenamiento. No aplica cuando usamos la función multinomial.

- `l1_ratio`: elastic-net es un tipo de regularización híbrida entre L1 y L2. Este parámetro fija el peso de L1 en dicha combinación. Nos es irrelevante, al usar L1.

LinearSVC:

- `penalty`: especifica el tipo de regularización a usar. Lo fijamos a 'l1' para usar regularización de tipo L1.
- `loss`: especifica la función de pérdida: `hinge` o `squared_hinge`. Solo `squared_hinge` permite usar L1, así que es la que seleccionamos.
- `dual`: indica si se usa la formulación dual o primal del problema de optimización. Puesto que tenemos más ejemplos que características, usaremos la formulación primal, es decir, dejamos `dual` a `False`.
- `tol`: tolerancia considerada como criterio de parada. Dejamos el valor por defecto, 0.0001, ya que disminuirlo podría hacer aumentar notablemente el tiempo de cómputo.
- `C`: como ya hemos visto, determina la intensidad de la regularización de forma inversamente proporcional. Se tomará $C \in \{0.1, 0.5, 1, 5, 10\}$.
- `multi_class`: determina la estrategia en problemas multiclase. Seleccionamos 'ovr' (*One vs. Rest*) puesto que la alternativa, el método de Cramer y Singer, tiende a obtener resultados peores con tiempos de ejecución mucho más elevados.
- `fit_intercept`: añade un término de sesgo. Lo activamos, puesto que no se lo hemos incluido manualmente.
- `intercept_scaling`: fija el valor del sesgo. Lo dejamos a 1.
- `class_weight`: asigna un peso a cada clase. Sirve como mecanismo contra el desbalanceo de clases. Como no es nuestro caso, lo dejamos al su valor por defecto (todas las clases con peso 1).
- `randomState`: semilla para los procesos pseudo-aleatorios. Le pasamos nuestra semilla prefijada.
- `max_iter`: máximo número de iteraciones del 'solver'. Consideramos 10000 iteraciones para que tenga suficientes para converger pero no necesite demasiado tiempo de cómputo.

Análisis de Resultados y Selección Final

A partir de la discusión realizada en los apartados anteriores, estamos en disposición de comparar cada modelo para los distintos valores de C que se han propuesto. Aplicaremos validación cruzada sobre las distintas variantes de RL y sobre las de SVM lineal por separado. Los resultados pueden verse en la tabla 1:

C	**Regresión Logística (SAGA, L1)**	**SVM Lineal (liblinear, L1)**
0.1	0.96	0.9373
0.5	0.9665	0.9398
1	0.9666	0.9403
5	0.9669	0.941
10	**0.9671**	0.9411

Table 1: Resultados de validación cruzada.

Vemos que los mejores resultados se dan, para ambos modelos, con la menor regularización probada ($C = 10$), aunque las diferencias son muy pequeñas. En este caso, la RL Multinomial parece obtener los mejores resultados, en lo que a tasa de acierto se refiere. En la figura 9 podemos ver las matrices de confusión de los resultados en validación-cruzada de los modelos tipo RL y SVM Lineal con menos errores. Una rápida comparación es suficiente para observar que ambos modelos fallan en casos similares (confundiendo las clases 4 y 7, ó la 1 y la 9), pero RL se equivoca menos que SVM Lineal de forma consistente.

La observación anterior es suficiente para **elegir RL** con $C = 10$ y una tasa de acierto en validación-cruzada de 0.9671 como nuestro modelo final.

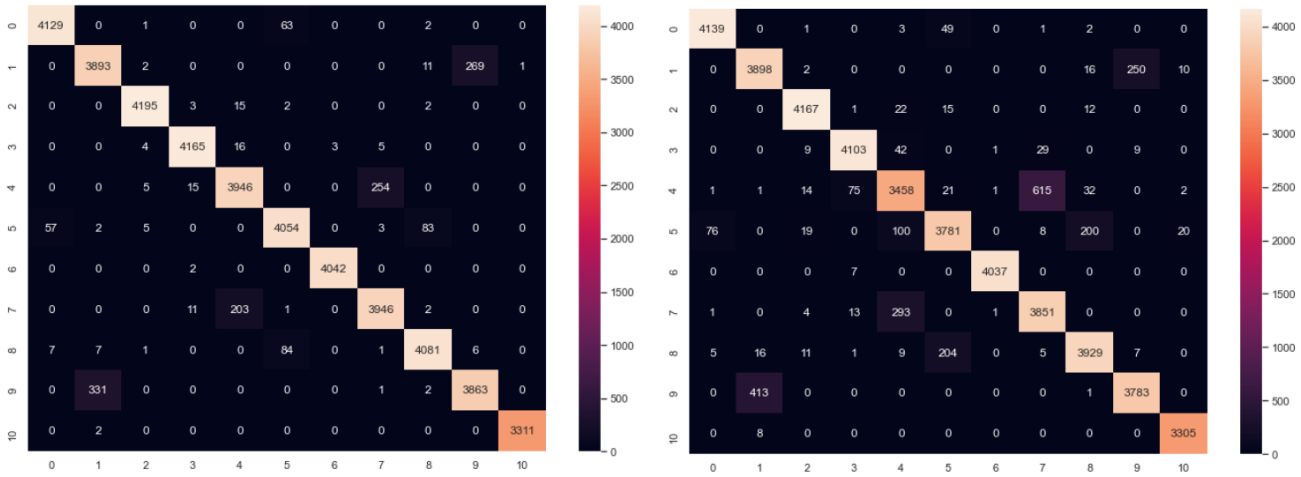


Figure 9: Matriz de confusión de los mejores modelos tipo Regresión Logística y SVM Lineal.

Test

Finalmente, reentrenamos nuestro modelo tipo Regresión Logística Multinomial con $C = 10$ usando todo $\mathcal{D}_{train}$. **La tasa de acierto de nuestras predicciones en test es:**

Tasa de acierto en test: 96.62%

La matriz de confusión del resultado final (figura 10) sigue en la línea de los resultados anteriores.

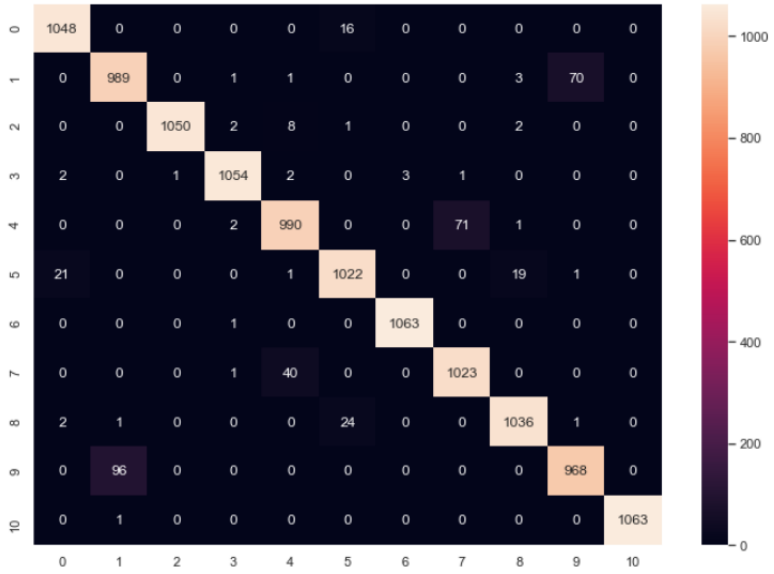


Figure 10: Matriz de confusión de las predicciones en test

Concluimos que no parece haber tenido lugar ningún tipo de sobreajuste, puesto que el resultado en test ha sido prácticamente el mismo que en validación.

2 Problema de Regresión

Descripción del problema

En esta apartado trabajaremos con un conjunto de datos correspondientes a propiedades de distintos superconductores, entre las cuales se encuentra su temperatura crítica, por debajo de la cual el material presenta resistencia nula al paso de electricidad. El dataset puede descargarse desde su repositorio en UCI, mientras que en este artículo se detallan aplicaciones de los superconductores, se analizan los datos y se describe un experimento en el que se construye un modelo estadístico que predice la temperatura crítica de un superconductor a partir del resto de características. Este problema de regresión es el mismo que trataremos de resolver nosotros, aunque nuestra forma de proceder será distinta puesto que nos restringiremos a modelos lineales.

Nuestros datos están etiquetados con un número real, la temperatura crítica de cada superconductor. Desde el punto de vista del Aprendizaje Automático, este es un problema de **regresión supervisada**, esto es, en el que queremos aproximara partir de un conjunto de datos etiquetados una función $f\colon \mathbb{R}^d \to \mathbb{R}$, donde d es el número de características de nuestro conjunto de datos, capaz de decirnos la temperatura crítica de un superconductor a partir del resto de propiedades consideradas.

Destacamos que algunas propiedades que toman valores discretos ordinales, como el número de elementos químicos que forman el material o el rango de valencia, se considerarán como variables reales.

Clases de funciones

Para este problema consideraremos dos modelos lineales para regresión sencillos: **regresión lineal** (o cualquiera de sus variantes con regularización, Lasso o Ridge, en caso de considerarlo oportuno) y **Support Vector Regression**, una variación de las SVMs para regresión que busca el hiperplano que minimice el ancho del "pasillo" que forman los datos en torno suyo, es decir, aquel para el que todos los datos, o la mayor parte de ellos, se encuentran lo más cerca posible del hiperplano aproximador (ejemplo en la figura 11). Esto garantiza una solución con menor dimensión VC que otros modelos lineales.

Por último, de forma adicional, incluiremos en nuestro estudio un modelo de SVR que haga uso de un kernel no lineal. Tal como se ha visto en teoría, esto implica que el algoritmo aplique implícitamente una transformación no lineal del espacio de características. El kernel que usaremos será de tipo *Radial Basis Functions* (RBF). Este kernel cuenta con la ventaja de mapear los ejemplos de entrenamiento a un espacio de dimensión infinita.

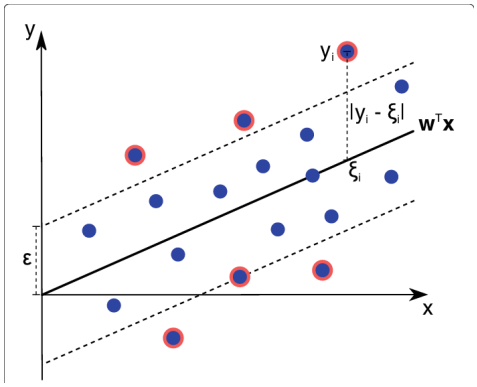


Figure 11: Ejemplo de SVR (fuente de la imagen).

Por otro lado, regresión lineal es un modelo sencillo y rápido de entrenar. En regresión es habitual utilizar el error cuadrático medio (ECM) como medida del error. El ECM no nos permite saber por sí solo como de bueno es un modelo, si no es en comparación con el ECM de otros modelos. Para completar la información aportada por ECM, contar con un modelo básico como regresión lineal puede ser útil como referencia para analizar los resultados de SVR.

Ahora debemos considerar la posibilidad de incluir **transformaciones no lineales** sobre nuestros datos. El argumento será el mismo que en el problema de clasificación.

En este caso, nuestro conjunto de datos cuenta con 81 características y 21263 instancias. El número total de características para $k = 1, 2, 3$ sería:

$$NTC(81, 1) = \binom{81 + 1 - 1}{1} = 81$$
$$NTC(81, 2) = 81 + \binom{81 + 2 - 1}{2} = 81 + 3321 = 3402$$
$$NTC(81, 3) = 3402 + \binom{81 + 3 - 1}{3} = 3402 + 91881 = 95283$$

En este caso, el número de variables crece incluso más rápido que en el problema de clasificación. Para este ejercicio contamos con menos datos de entrenamiento que antes, lo que podría acelerar el aprendizaje. Tomar $k = 3$ queda descartado, puesto que el número de variables sería muy superior al de ejemplos, haciendo el aprendizaje más complejo y mucho más lento, a no ser que nos restringiésemos a formulaciones duales del problema. $k = 2$ aumenta notablemente la dimensionalidad de nuestros datos, pero aún por debajo del número de instancias. Seguiremos el mismo procedimiento que en problema de clasificación y extenderemos nuestro dataset con combinaciones cuadráticas de las características originales.

De nuevo, descartamos la opción de aumentar mucho el número de características y restringirnos al problema dual porque 20000 instancias hacen computacionalmente inviable su resolución.

Análisis Exploratorio y Preprocesamiento

Lectura y Separación de los datos

Nuestros datos son leídos de forma conjunta desde el fichero `superconduct/train.csv`. Comprobamos que no hay valores perdidos, y separamos los datos y sus etiquetas en estructuras de datos separadas. Por último, separamos los datos en $\mathcal{D}_{test}$ y $\mathcal{D}_{train}$ de igual forma que se hizo para clasificación.

Análisis descriptivo y selección de características

Ya se justificó para clasificación la importancia de saber si nuestros datos seguían una distribución normal. No se utilizaron argumentos exclusivos a la tarea de clasificar, sino que esto nos da una información descriptiva de la muestra generalmente útil, y aplican de igual manera al caso actual.

El test de normalidad de SciPy nos informa de que, casi con toda seguridad (p-valor menor a $3 \cdot 10^{-55}$), rechazamos la hipótesis nula y nuestros datos no siguen una distribución normal.

Tras analizar nuestras características originales, aplicamos las **transformaciones cuadráticas** establecidas en la sección anterior. De nuevo, repetimos las mismas transformaciones sobre el conjunto de test, puesto que el modelo lineal que entrenaremos ahora asumirá un dato del espacio de características transformado.

A continuación repetiremos el análisis de colinealidad, esta vez entre las variables de nuestro nuevo dataset transformado.

La matriz de coeficientes de correlación de Pearson está representada visualmente en la figura 12 antes y después de eliminar variables redundantes siguiendo el mismo procedimiento que en el problema de clasificación. Vemos que el número de variables decrece significativamente, de más de 3000 a menos de 350.

Ahora vamos a repetir el mismo análisis de la información mutua entre las distintas variables, observadas individualmente, y el vector de etiquetas. La figura 13 muestra el resultado, y algo salta a la vista en comparación con lo encontrado anteriormente para clasificación: todas las variables tienen una información mutua relativamente alta, al menos si las comparamos entre sí. Decidimos no eliminar ninguna variable de acuerdo a su índice de información mutua.

La selección de características realizada nos deja con un total de 349 variables. Es importante seleccionar en el conjunto de test las mismas variables que han resultado de este procedimiento, puesto que son las que nuestros modelos aprenderán a interpretar.

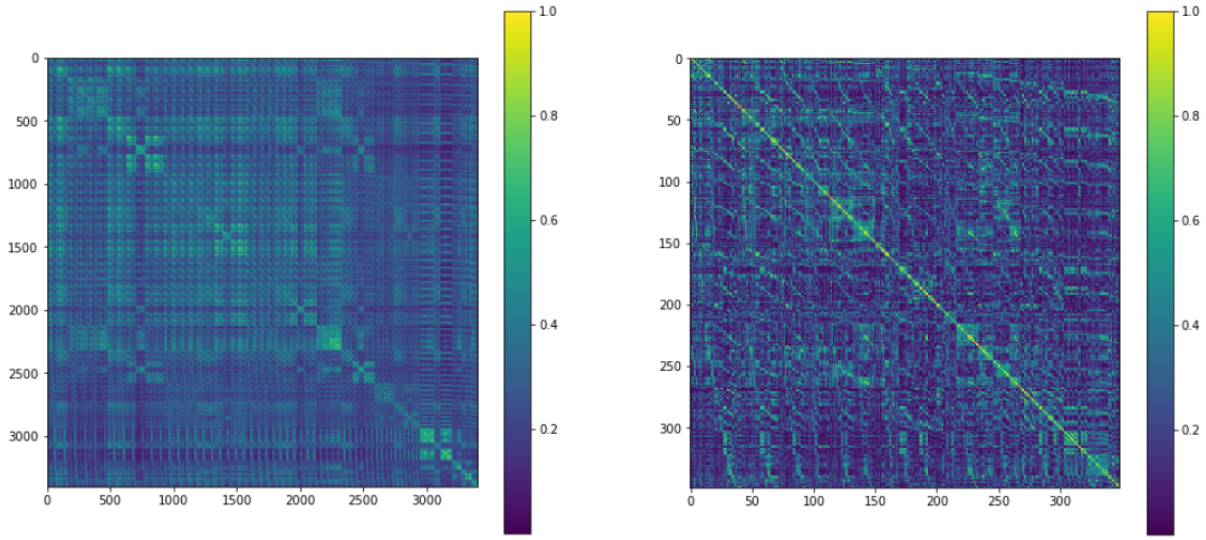


Figure 12: Correlaciones lineales entre características antes (izquierda) y después (derecha) de eliminar variables redundantes.

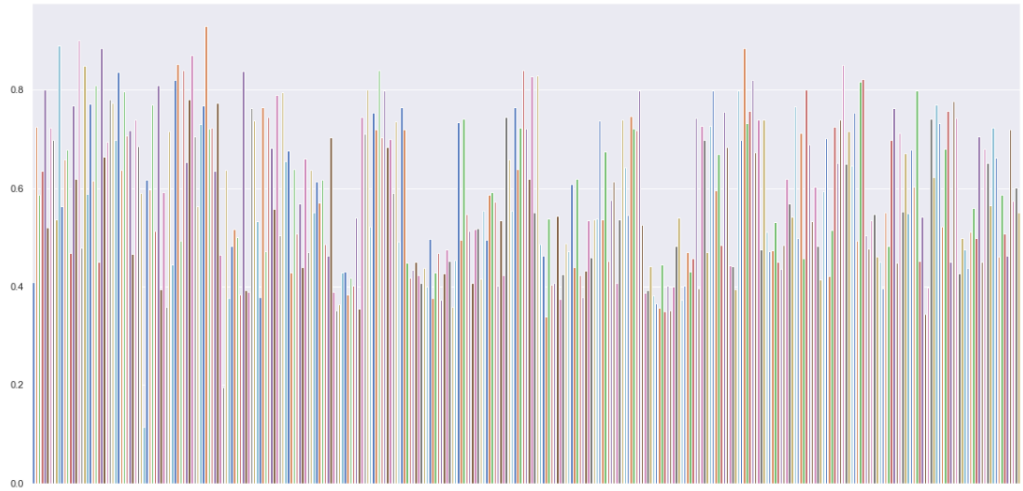


Figure 13: Información mutua de cada variable con el vector de etiquetas.

Normalización

Vamos a analizar si es necesario normalizar nuestro conjunto de datos en este caso. La figura 14 nos muestra las medias y desviaciones típicas de los datos en $\mathcal{D}_{train}$, con las características restantes después de la selección del apartado anterior. Nótese que la escala es logarítmica.

La figura 14 deja patente la diferencia en escala entre las distintas variables. Normalizar nuestro conjunto de datos es obligatorio. Además, a la hora de visualizar nuestras variables en busca de *outliers*, normalizar será sumamente conveniente.

De nuevo, normalizamos restándole a cada variable su media y dividiendo el resultado por su desviación típica. Como antes, transformación aplicada sobre el conjunto de entrenamiento debe replicarse sobre el de test.

La figura 15 muestra un diagrama de cajas con las variables estandarizadas. No parece haber presentes outliers flagrantes, al menos observando cada variable individualmente, así que en este caso no eliminaremos valores extremos.

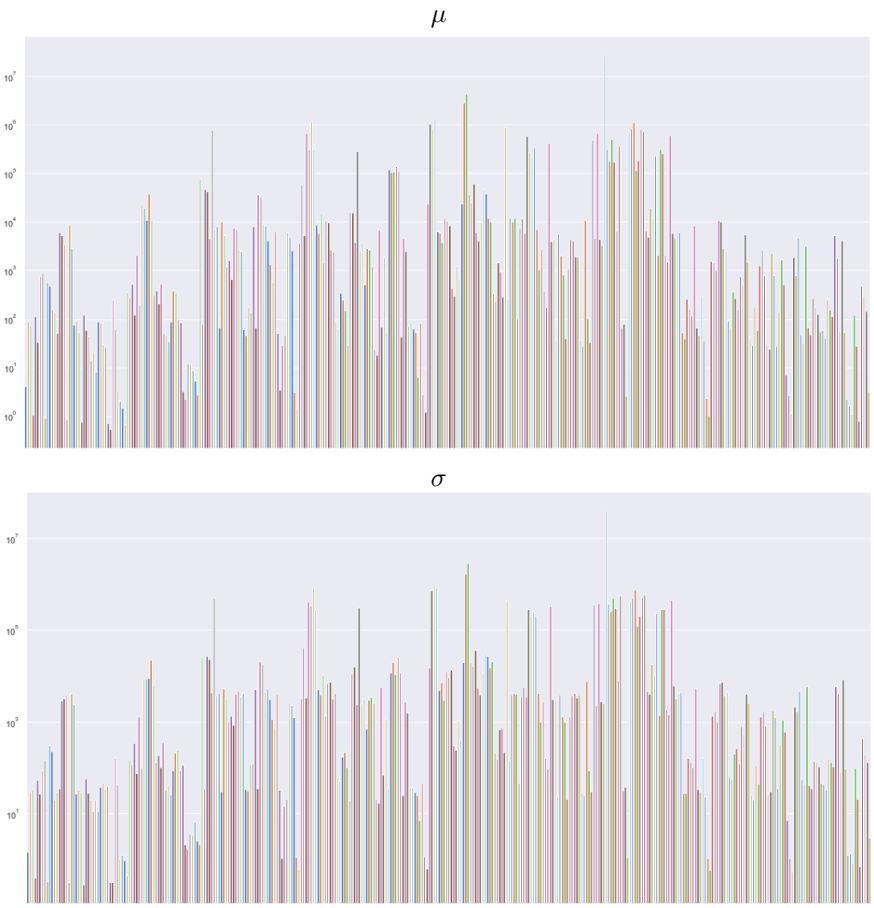


Figure 14: Medias (arriba) y desviaciones típicas (abajo) en escala logarítmica de nuestro conjunto de entrenamiento

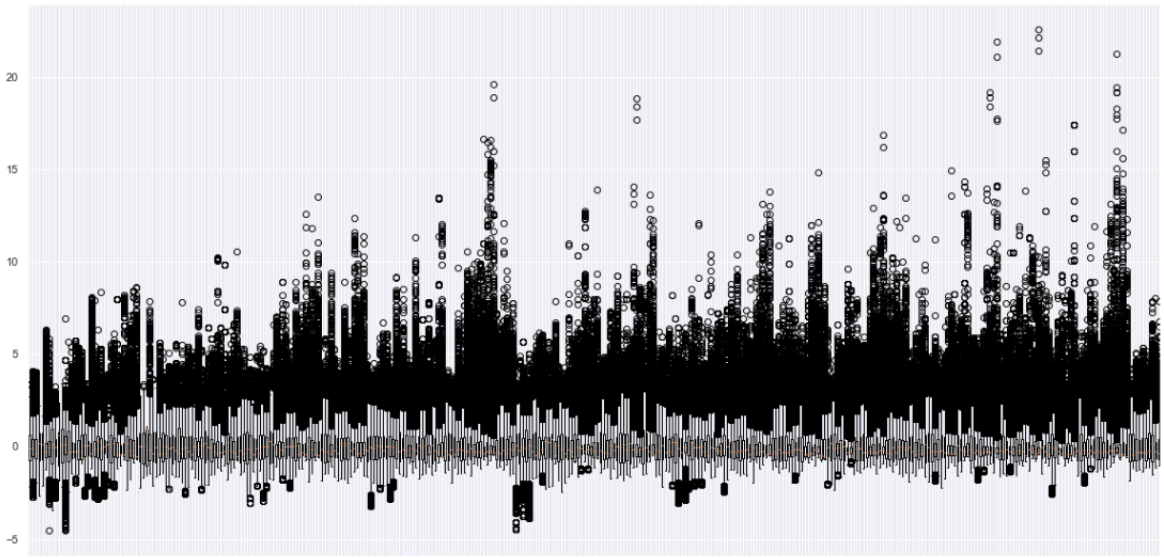


Figure 15: Diagrama de cajas de las variables de entrenamiento tras normalizar.

Selección del Modelo final

El proceso de selección de modelos será análogo al del problema de clasificación.

K-fold y Métrica del Error

El número de *folds* para validación cruzada se ha fijado a 5 por los mismos motivos que en clasificación.

En este caso hemos elegido **dos métricas** del error: **ECM** y $\boldsymbol{R^2}$ **score**. El error cuadrático medio es una medida intuitiva que resulta natural para el problema de regresión. Tiene la propiedad de penalizar más los errores grandes, y despenalizar los pequeños. El coeficiente R^2 determina "la calidad del modelo para replicar los resultados, y la proporción de variación de los resultados que puede explicarse por el modelo". Toma valores en $[-\infty, 1]$, donde un 1 es la mejor puntuación posible. Un modelo que siempre devolviese el valor esperado de la etiqueta obtendría una puntuación de 0. Esta medida es, en cierto sentido, más absoluta que el ECM al permitir una interpretación sin necesidad de compararla con los resultados de otros modelos, por lo que puede resultar beneficioso a la hora de analizar los resultados.

Regularización

Nuestra elección de la regularización se fundamentará en el análisis exploratorio de los datos y en el preprocesamiento realizado en secciones anteriores. Consideraremos dos tipos de regularización, L1 (o LASSO) y L2 (o Ridge).

En este caso, hemos visto que todas las variables poseen cierta "importancia" en la predicción de la etiqueta, o así nos lo indican los índices de información mutua entre las características y el vector de etiquetas. Como consecuencia, difícilmente podríamos sacar partido a una regularización tipo Lasso, que tiende a eliminar características inútiles. Por otro lado, sabemos que L2 tiende a mitigar las colinealidades. En nuestro caso ya hemos eliminado aquellas variables con correlaciones muy elevadas, aunque esto se ha hecho fijando un umbral relativamente alto (de 0.9). Además, nuestro análisis de correlaciones es componente a componente, mientras que una regularización de este tipo podría ayudar a reducir el efecto de relaciones de multicolinealidad.

Es por esto que consideraremos regularización tipo L2. No esperamos, sin embargo, que esta regularización juegue un papel importante porque ya hemos eliminado muchas de las variables redundantes. Es por esto que reduciremos la intensidad de regularización. Probaremos con valores de C en $\{1, 5, 10, 15, 20\}$ (recordamos que C es inversamente proporcional a la fuerza de regularización).

Otro mecanismo de regularización presente en los SVR es el valor de ϵ. Este parámetro define el margen que se le da a los puntos fuera del "pasillo" en torno al hiperplano aproximador para no contar como errores (ver figura 11). Probaremos con $\epsilon \in \{0.0, 0.1, 0.5, 1.0\}$.

En lo que respecta al modelo extra no lineal con el que nos queremos comparar, los kernels de base radial cuentan con un parámetro γ que ejerce un papel regularizador. En particular, este es inversamente proporcional al radio de las bases gaussianas. Un γ bajo generará soluciones más suaves y uno alto soluciones tan complejas como se quiera.

Otros Parámetros del Entrenamiento

Decidimos usar las clases `LinearRegression`, `LinearSVR` y `SVR` (para el kernel no lineal) de Scikit-Learn para implementar nuestros modelos. Procedemos a explicar los parámetros de las tres clases:

Regresión Lineal:

- `fit_intercept`: añade un término de sesgo. Lo activamos, puesto que no se lo hemos incluido manualmente.
- `normalize`: indica si se desea normalizar los datos antes del aprendizaje. Lo fijamos a `False`.
- `copy_X`: indica si copiar el conjunto de datos antes de iniciar el proceso. En caso contrario podría sobrescribirse. Lo dejamos activado.
- `n_jobs`: número de hilos en paralelo que usar para el cálculo. Lo fijamos a 6.

LinearSVR:

- `epsilon`: margen de error aceptado, explicado en el apartado anterior.
- `tol`: tolerancia considerada como criterio de parada. Dejamos el valor por defecto, 0.0001, ya que disminuirlo podría hacer aumentar notablemente el tiempo de cómputo.
- `C`: como ya hemos visto, determina la intensidad de la regularización de forma inversamente proporcional. Se tomará $\{1, 5, 10, 15, 20\}$.
- `loss`: especifica la función de pérdida, "epsilon_insensitive" o "squared_epsilon_insensitive". Seleccionamos "squared_epsilon_insensitive" al corresponder con la regularización L2.
- `dual`: indica si se usa la formulación dual o primal del problema de optimización. Puesto que tenemos más ejemplos que características, usaremos la formulación primal, es decir, dejamos `dual` a `False`.
- `fit_intercept`: añade un término de sesgo. Lo activamos, puesto que no se lo hemos incluido manualmente.
- `intercept_scaling`: fija el valor del sesgo. Lo dejamos a 1.
- `randomState`: semilla para los procesos pseudo-aleatorios. Le pasamos nuestra semilla prefijada.
- `max_iter`: máximo número de iteraciones del 'solver'. Consideramos 10000 iteraciones para que tenga suficientes para converger pero no necesite demasiado tiempo de cómputo.

`SVR` tiene parámetros muy similares a los de `LinearSVR`, con las siguientes diferencias:

- No permite especificar la función de pérdida. Se usa la formulación dual con el kernel que se le especifique (en este caso, 'rbf') optimizando la función que aparece aquí. Esta incluye regularización de tipo L2.
- Un parámetro `gamma`, que fija el valor del coeficiente de regularización explicado en el apartado anterior. Dejaremos el valor por defecto, 1/(n_features*X.var()), puesto que es conveniente tomarlo inversamente proporcional a la varianza del conjunto de datos. Intuitivamente, cuanto más dispersos se encuentren los datos mayor tendrá que ser el radio de las funciones de base radial.

Análisis de Resultados y Selección Final

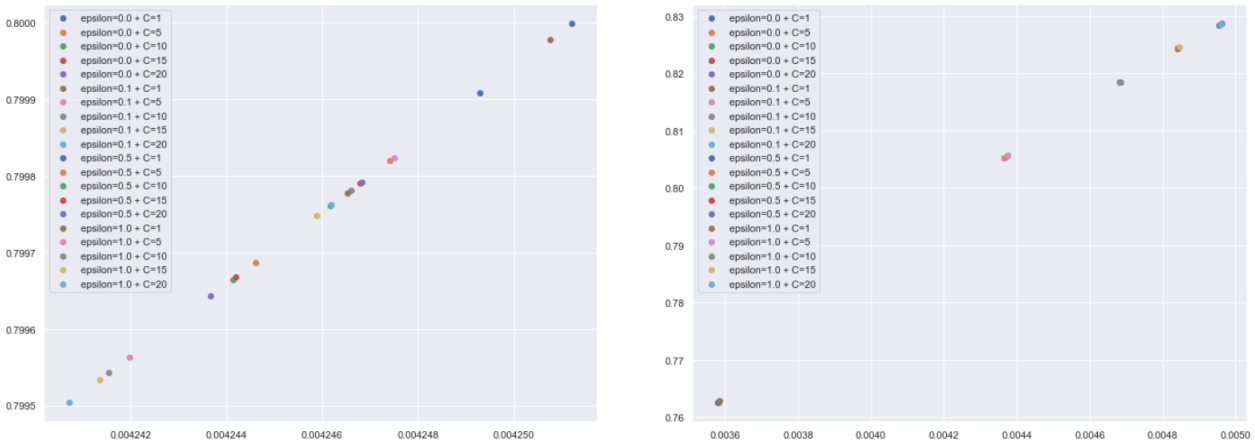
A partir de la discusión realizada en los apartados anteriores, estamos en disposición de comparar cada modelo para los distintos valores de C y de ϵ que se han propuesto. Aplicaremos validación cruzada sobre Regresión Lineal y sobre las distintas variantes de SVR.

Ahora nos encontramos con una complicación añadida. Hemos elegido dos métricas distintas para comparar resultados, que no tienen por qué elegir conjuntamente al mismo ganador. En la figura 16 vemos que, en este caso, aquellos resultados con menos ECM son los que obtienen mayores R^2, con lo que podemos usar cualquiera de las dos métricas como criterio para elegir el mejor modelo.

Comparemos los resultados obtenidos por Regresión Lineal, SVR Lineal y SVR RBF. La tabla 2 muestra los mejores resultados obtenidos por cada clase de funciones.

	ECM	$\boldsymbol{R^2}$
Regresión Lineal	235.527	0.799743
SVR Lineal $\boldsymbol{\epsilon = 0.0}$**, C=1**	235.22685	0.799998
SVR RBF $\boldsymbol{\epsilon = 0.1}$**, C=20**	201.534116	0.828645

Los resultados de SVR lineal mejoran muy sutilmente los de regresión lineal (solo hay diferencias en sus índices R^2 a partir de la cuarta cifra decimal). Esto podría hacernos plantearnos si, con un conjunto de datos más grande con el que llevase mucho tiempo entrenar, regresión lineal podría ser una opción viable para obtener resultados aceptables en un tiempo más asequible. También es un indicador de que el factor que más ha podido influir para obtener estos resultados sea el preprocesamiento, puesto que los modelos lineales no logran diferencias significativas entre ellos (no hay más que ver la escala de la figura 16). La poca diferencia entre Regresión Lineal (sin regularización) y SVR Lineal (con regularización) también puede ser un indicador que



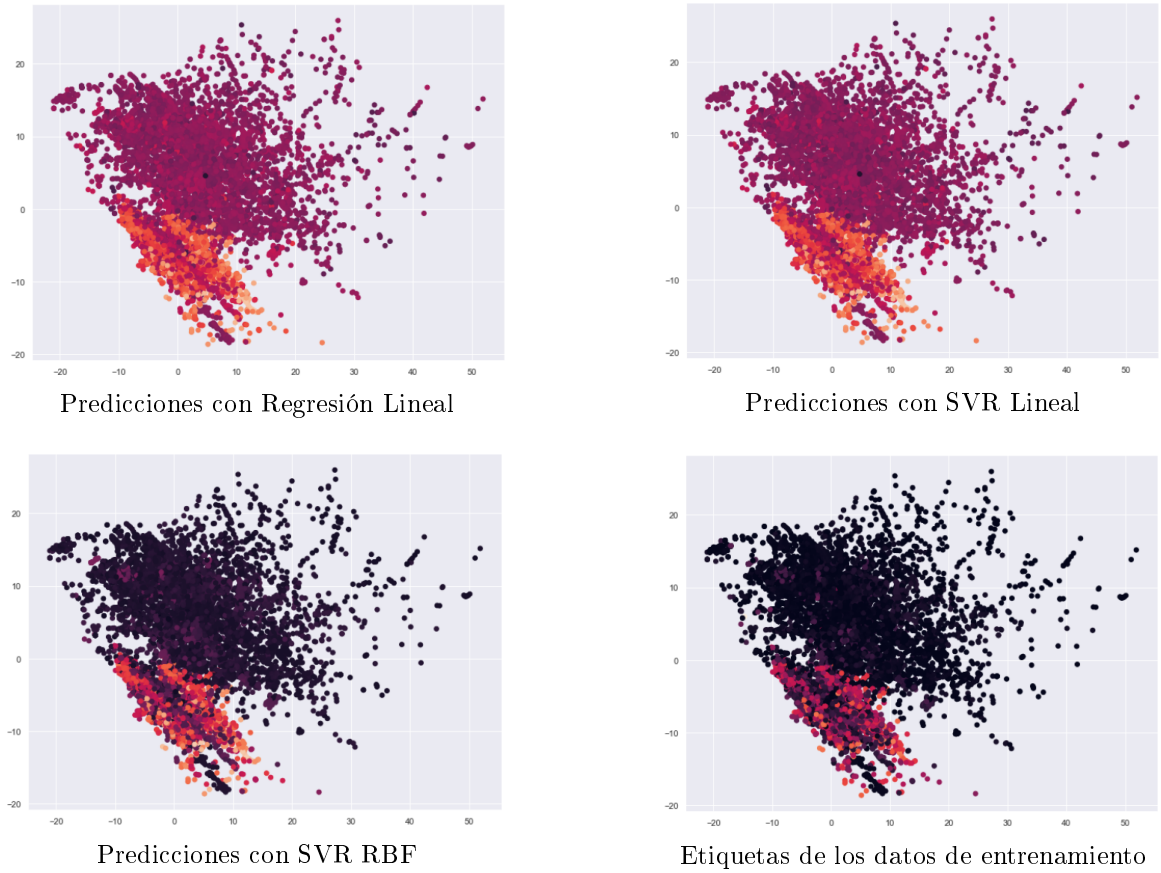
Kernel lineal

Kernel RBF

Figure 16: Representación de 1/ECM (eje x) frente a los índices R^2

confirme nuestra sospecha de que la regularización no iba a ser de gran importancia en a la hora de encontrar una solución adecuada y suficientemente general al problema.

Los resultados de SVR RBF contituyen mejoras significativas sobre los de los modelos lineales. En la figura 17 mostramos una proyección bidimensional de los datos de entrenamiento creada mediante PCA. La intensidad del color de cada punto representa el valor de la función (temperatura crítica) que tiene asociado. No debemos pensar que esta es una representación fiel de los datos, puesto que las dos primeras componentes principales solo logran explicar un 32% de la varianza. Sin embargo, sí podemos destacar el cambio brusco que la función toma en torno a los ejemplos que vemos abajo en naranja. Claramente, los modelos lineales no logran replicar estos cambios bruscos, y por eso el modelo no lineal logra resultados mucho mejores.



Predicciones con Regresión Lineal

Predicciones con SVR Lineal

Predicciones con SVR RBF

Etiquetas de los datos de entrenamiento

Figure 17: Proyección mediante PCA de los datos con sus etiquetas predichas y originales.

Test

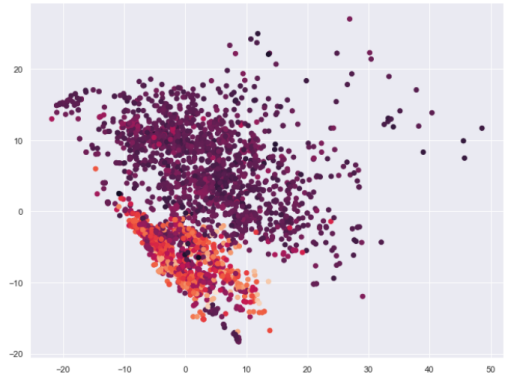
Finalmente, vamos a comparar los resultados en test de SVR Lineal con $\epsilon = 0.0$ y $C = 1.0$ (modelo lineal con mejores resultados) con los del SVR RBF que ha sido vencedor en la selección de modelos ($\epsilon = 0.1$ y $C = 20.0$). **Nuestros resultados en test son:**

SVR Lineal:
ECR de SVR Lineal en test: 241.2125
R^2 de SVR Lineal en test: 0.79241

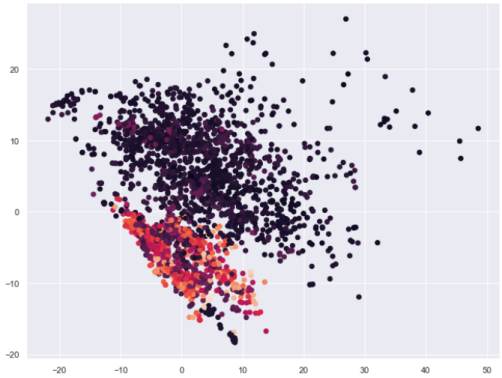
SVR RBF:
ECR de SVR RBF en test: 205.39251
R^2 de SVR RBF en test: 0.82324

Los resultados en test son ligeramente peores a los de validación. Esto es natural, puesto que hemos elegido aquellos modelos que mejor funcionaban en validación, y puesto que puede existir cierta variabilidad entre los datos de train y test. Sin embargo estas diferencias no son muy elevadas. El coeficiente R^2 en test no difiere de su análogo en vaidación hasta la tercera cifra decimal

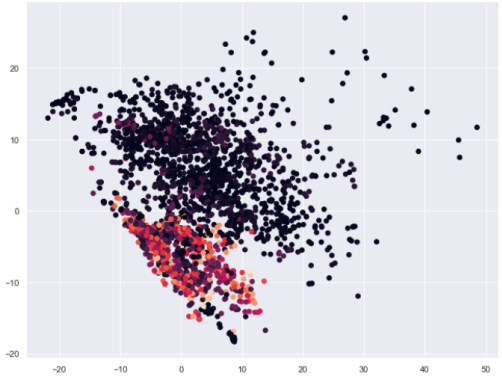
En la figura 18 vemos los resultados de los dos modelos seleccionados, ambos en la misma línea seguida en entrenamiento. El modelo lineal asigna etiquetas mucho más homogéneas a todos los puntos que el no lineal, lo que es una fuente de error en este caso.



Predicciones con SVR Lineal



Predicciones con SVR RBF



Etiquetas de test

Figure 18: Proyección mediante PCA de los datos de test con sus etiquetas predichas y originales.

Referencias

Clasificación

- *Learning From Data*, Y.S.Abu-Mostafa *et al.*
- Repositorio del dataset para clasificación.
- Artículo en el que se describe el dataset para clasificación.
- Scikit-Learn: Regresión Logística.
- Scikit-Learn: SVM con kernel lineal.
- Comparación entre RL Multinomial y OvR.
- Artículo introductorio a las SVMs.
- `www.machinelearningmastery.com` (Blog sobre Aprendizaje Automático).
- Análisis Discriminante Lineal.
- Guía para interpretar los coeficientes de correlación.
- Fundamento teórico para el coeficiente de información mutua entre variables continuas y discretas.
- *The Elements of Statistical Learning.*
- *Método del codo.*
- An Introduction to Statistical Learning.
- Comparación entre normalización en rango y enstandarización.
- Otra comparación entre normalización en rango y enstandarización.
- Normalización para SVMs.
- Regularización LASSO.
- Regularización Ridge.
- SAGA Solver.
- LIBLINEAR Solver.
- Análisis teórico del descenso de coordenadas.
- Comparación de diversos métodos multiclase para SVM.

Regresión

- Teoría sobre superconductores.
- Repositorio de UCI con el dataset utilizado.
- Estudio hecho sobre los mismos datos.
- Scikit-Learn: Regresión Lineal
- Scikit-Learn: Support Vector Regression
- Kernels de funciones de base radial (RBF).
- Fuente de la imagen de ejemplo sobre SVR (figura 11).
- Coeficiente R^2.
- Scikit-Learn: coeficiente R^2.