Práctica 3

Blanca Cano Camarero

Curso 2020-2021

Índice

Regresión. Problema de los superconductores

Nota preliminar

Para poder leer los datos, estos deben de estar situados en una carpeta con la siguiete estructura:

```
codigo/
|-- clasificacion.py
|---regresion.py
|---datos
    |-Sensorless_drive_diagnosis.txt
    |-train.csv
    |-unique_m.csv
```

Descripción del problema

Se tienen dos ficheros que contienen 21263 datos sobre superconductores y sus características relevantes.

- Características de los ficheros: Multivariante
- Atributos: reales
- Tarea de regresión
- Número de instancias: 21263
- Número de atributos: 81
- No faltan valores.
- Área de físicas.

Se pretende predecir el punto crítico de ruptura.
De las características obtenidas

("UCI Superconductivty Data Data Set" n.d.)

Tratamiento de los datos

Trabajaremos solo con el primer fichero, ya que el segundo contiene los compuestos químicos de los que hemos extraído los datos y no nos es relevante. (Hamidieh 2018)

Distribución de las etiquetas de entrenamiento

Procederemos con un análisis preliminar de las etiquetas, para ver cuál es su distribuición. Se realizará con la función propia `BalanceadoRegresion(y, divisiones = 20)`.

Obtenemos los siguiente información relevante:
- Los valores se toman en el intervalo $[2 \times 10^{-4}, 185]$
- Mediana etiquetas: 20
- Media etiquetas: 34.4212
- Desviación típica de las etiquetas: 34.2536
- Media de número de etiquetas por intervalo: 708.7667
- Desviación típica de número de etiquetas por intervalo: 1136.9938

A vista de estos resultdos es notable que los valores no están repartidos homogéneamente, es más, presentan un fuerte desequilibrio con mayor presencia de etiquetas bajas y ausencia de etiquetas en ciertos intervalos altos.

Esto queda reflejado incluso cuando analizamos la dispersión en diferentes rangos,

Tabla 1: Distribución de la etiquetas con valor en cierto rango.

Intervalo a analizar:	$[2 \times 10^{-4}, 185]$	$[100, 185]$	$[143, 185]$
Número total de etiquetas	21263	768	3
Mediana de las etiquetas	20	112	143
Número de etiquetas medio por partición	708.7667	25.6	0.1
Desviación típica de cantidad de etiquetas en intervalo	1136.9938	35.83	0.3958

Estrategias ante esta distribución No podemos ampliar la muestra, así que la única opción para conseguir más homogeneidad en las etiquetas sería descartar ciertos datos; como esto nos haría perder precisión en general optaremos por utilizar los datos que tenemos, siendo conscientes de que el entrenamiento para valores mayores es peor.

Tipificación de los datos

Procederemos a tificar los datos. Esto nos va a dar algunas ventajas como reducier la gran diferencia de escala en los valores manteniendo las difrencias.

Exiten diferentes métodos de transformación (z-score, min-max, logística...), nosotros hemos optado por el Z-score. ("Sobre normalización En Aprendizaje Automático" n.d.) Que consiste en una transformación de la variable aleatoria X a otra, Z de media cero y varianza uno.

$$Z = \frac{x - \bar{x}}{\sigma}$$

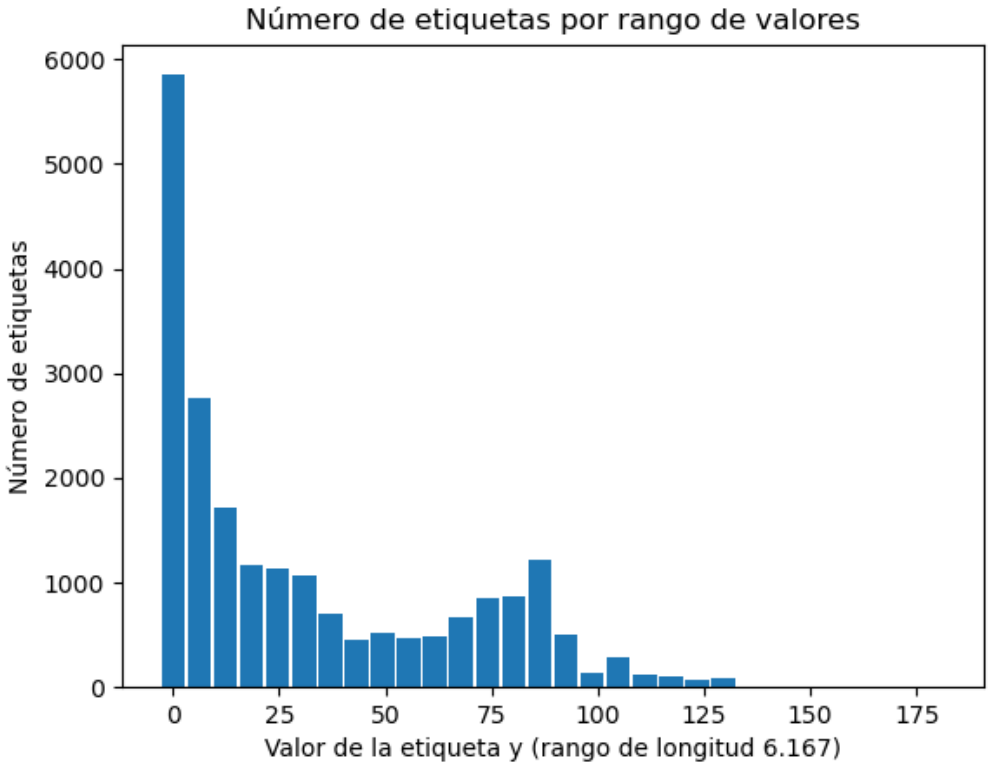
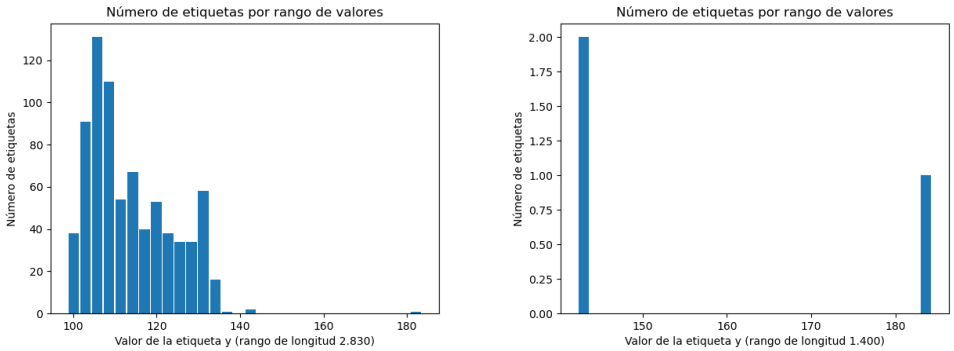


Figura 1: Distribución de las etiquetas en rango $[2 \times 10^{-4}, 185]$



(a) Distribución de las etiquetas en rango $[100, 185]$

(b) Distribución de las etiquetas en rango $[143, 185]$

Donde $\bar{x}$ representa la media de X y σ la desviación típica.

Para la implementación utilizamos la función `StandardScaler()` y los métodos `fit_transform( x_train )` y `scaler.transform( x_test)`. ("SrandardScaler Del

Paquete sklearnPreprocessing" n.d.)

La necesidad de estos método es normalizar a partir de los datos de entrenamiento, guardar la media y varianza de estos datos y luego aplicar la misma transformación (con los mismo datos de entrenamiento) al test, esto se realiza así ya que si se aplicara la transformación a todos los datos se estaría cometiendo data snopping.

Reducción de la dimensión

A continuación intentaremos reducir el tamaño de vector de características sin perder explicación en los datos.

Para ello utilizaremos el coefiente de correlación de Pearson, que se define como sigue:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_X \sigma_Y}$$

Donde:

- X, Y son dos variables aleatorias que siguen la misma distribución, en nustro caso dos características distintas.
- cov la covarianza.
- σ_X la desviación típica de X.

La interpretación es la siguiente, 1 si existe una correlación perfecta, -1 si la correlación inversa es perfecta y cero si no existe relación alguna entre las características.

La matriz de correlación de los datos queda reflejada en figura 3.

Los datos explicados a partir de otros son los que se aproximan a uno (blanco) o a menos uno (negros) y estos son los que eliminaremos.

Para tener una visión más analítica de los resultados utilizaremos la función `Person(x,umbral, traza)`, esta nos indicará qué características están relacionadas, con coeficientes en valor absoluto mayor que umbral indicado.

La mayor correlación empieza a partir de 0,9977, relaciones con correlación superior a 0,99 hay 5.

A continuación muestro una tabla que refleja cómo variaría la dimensión de nuestros datos si eliminamos una de las columnas que sea explicada por otra con un umbrar superior al indicado.

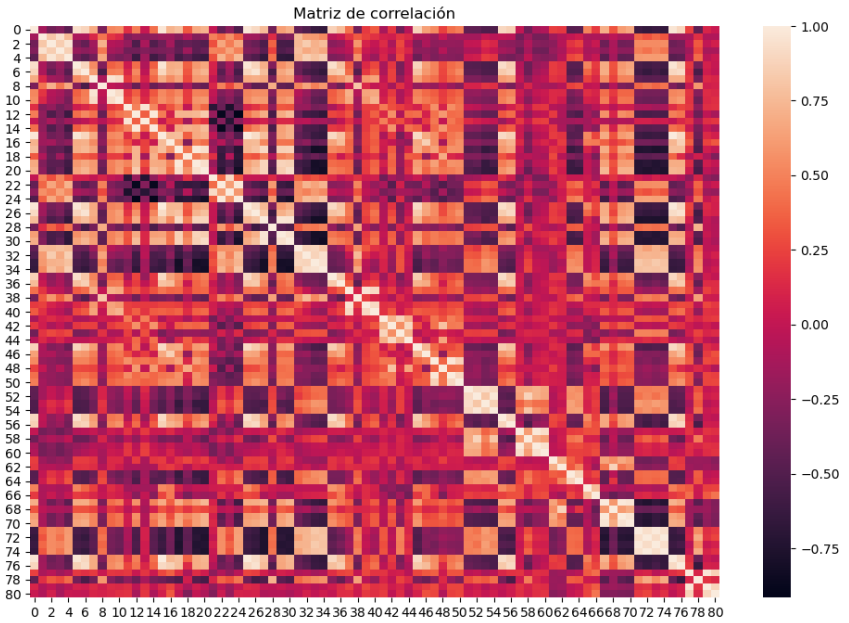


Figura 3: Matriz de correlación de los datos

Tabla 2: Reducción de la dimensión de la matriz de características a partir del umbral de coeficiente de correlación indicado

umbral	tamaño tras reducción	reducción total
0.999	81	0
0.99	76	5
0.98	72	9
0.97	68	13
0.95	58	23
0.9	42	39

Puede consultar los coeficientes respectivos durante la ejecución del código, aquí le muestro para un umbral de 0.97.

Tabla 3: Coeficiente pearson para umbral 0.97

Coeficiente	Índice 1	Índice 2
0.9977284401815567	15	25
0.9949859517136572	72	74
0.9927038888466977	15	75
0.9922969155759501	12	14
0.9898601215727296	71	73
0.9894939628750227	25	75
0.987794792442112	67	69
0.9847899736486817	57	59
0.9815600785302888	17	19
0.980149145006249	22	24
0.9738111111636902	77	79
0.9732998811054523	47	49
0.9731669940410288	0	15
0.9722391366986369	0	25

- score: devuelve la función de verosimilitud logarítmica logarítmica, es decir cuánto de buenos es el ajuste, cuanto mayor sea el ajuste mejro será. https://en.wikipedia.org/wiki/Likelihood_function https://medium.com/(**analyttica/log-likelihood-analyttica-function-series-cb059e0d379?**) Cuanto más alto sea mejor, no tiene cota así que calcularemos uno sin reducir la dimensión para tenerlo como cota, los resultado obtenidos son:

Tabla 4: PCA y su máxima verosimilitud

N componentes	score sin haber reducido	score habiendo reducido
1	-97.49	-83.35
2	-91.81	-78.77
34	-2.557	-13.03
51	9.146	-6.244
72	17.9	-6.244
68	16.53	-3.905
81	19.95	-3.905

Además es interesante comparar el valor 68, con experimentos posteriores, ya que es el número de dimensiones al que se redució usando el coeficiente de pearson.

Además de manera general reducir dimensiones emperora drásticamente la verosimilitud, sobretodo si no hay variables redundantes (nótese el caso en el que se habían reducido previamente las dimensiones con Pearson).

Aprovechando que hemos calculado una con dimensión uno vamos a visualizar los datos:

Esta forma nos recuerda a un función de proporcionalidad inversa, quizás sea interesante en estadios posteriores probar con una transformación de este tipo.

Error a utilizar

Si bien en los errores hemos utilizado en clase el error cuadrático medio, *mean_squared_error* (**Varianza?** explicada). El penaliza más a las grandes diferencias.

Hemos optado por utilizar R^2, el coeficiente de determinación, que no es más que el coeficiente de correlación de Pearson al cuadrado; ya que no dará un valor acotado en el intervalo $[0, 1]$, a diferencia del error cuadrático medio que no lo está.

A nivel computacional puede que en algunos caso por la forma de calcularlo el coeficiente sea negativo, esto se redondeará a cero.

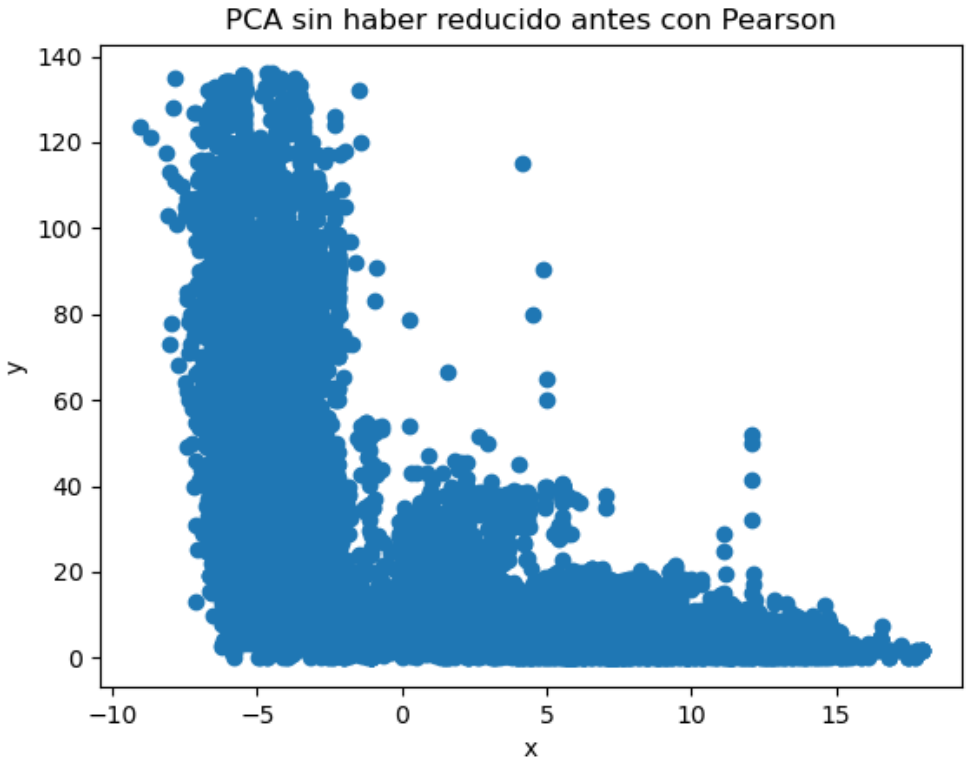


Figura 4: Visualización de reducción a una dimensión de las etiquetas

Selección del modelos

Modelos que vamos a tratar

La infomación a todos estos modelos ha sido sacada de la página oficial de skleart entre el 23 de mayo de 2021 y e 5 de julio del mismo año.

Modelo de regresión lineal

Se corresponde a un modelos de regresión lineal por mínimos cuadrados usual, la funcion es `class sklearn.linear_model.LinearRegression(*, fit_intercept=True, normalize=False, copy_X=True, n_jobs=None, positive=False)`

De manera general el modelo trata de ajustar el vector de pesos minimizando el la suma de al cuadrado de la diferencia entre las etiquetas y el valor actual.

Este método tiene como característica que da más importancia a reducir grandes diferencias entre los datos.

.

Las variantes que posteriormente utilizaremos de este método son Lasso y Ridge, ambos introduce regularización.

Ridge

Regresión lineal de mínimos cuadrados con regularización l2, es decir la función objetivo a minimzar es:

$$f(w) = \|y - wX\| + alpha\|w\|^2$$

Donde $alpha \in \mathbb{R}^+$ es la fuerza de la regularización. Se corresponde al argumento `alpha`.

Este tipo de regresiones consiguen que los coeficientes tomen de manera general valores más bajos.

Es equivalente a usar `SGDRegresor` con los argumentos `SGDRegressor(loss='squared_loss', penalty='l2')`

Lasso

Regresión lineal de mínimos cuadrados con regularización l2, es decir la función objetivo a minimzar es:

$$f(w) = \|y - wX\| + alpha\|w\|$$

Donde $alpha \in \mathbb{R}^+$ es la fuerza de la regularización. Se corresponde al argumento `alpha`.

Este tipo de regresiones consiguen que los coeficientes tomen de manera general valores más bajos.

Es equivalente a usar `SGDRegresor` con los argumentos `SGDRegressor(loss='squared_loss', penalty='l1')`

Gradiente descendente estocástico

Implementa el gradiente descendiente estocástico, en la documentación se recomienda utilizar estos métodos cuando el número de valores sea lo suficientemente grande (mayor que 10000). En nuestro caso contamos con más de 20000 tras el procesado, luego es legítimo su uso.

Las funciones de pérdida a minimizar pueden ser:

- `loss="squared_loss"`: Ordinary least squares,
- `loss="huber"`: Huber loss for robust regression,
- `loss=.epsilon_insensitive"`: linear Support Vector Regression.

Nosotros utilizaremos la de mínimos cuadrados y la última.

La mayor ventaja de este método es su eficiencia computacional $\mathcal{O}(kn\bar{p})$ donde k es el número de épocas y $\bar{p}$ es la media de los atributos no nulos del conjunto, n es el número de características de la matriz de entrenamiento $X\mathbb{R}^{n\times p}$.

Epsilon-Insensitive

Esto es un caso de soft-margin equivalente a regresiónd de soporte vectorial donde la función a minimizar es:

$$L(y_i, f(x_i)) = max(0, |y_i, f(x_i)| - \epsilon)$$

La regla de actualización de los pesons viene dada por

$$w_{new} = w - \eta(\alpha\frac{\partial R(w)}{\partial w} + \frac{\partial L(w^T x_i + b, y_i)}{\partial w})$$

Donde η es la tasa de aprendizaje, η la tasa de aprendizaje que puede ser constante o gradualemnte menos usando `learning_rate = 'optimal'` R es el cuadrado de la norma euclídea en L2 o la norma uno en L1. vendría dada por

$$\eta^(t) = \frac{1}{\alpha(t_0 + t)}$$

Argumentos y parámetros generalel a todos los modelos

Todos los modelos nos permiten agilizar el ajuste utilizando programación en paralelo, esto se hace gracias `n_jobs`.

El error a usar será R^2.

Experimento transformaciones características

Vamos a seleccionar con qué transformación de los datos de entrenamiento vamos a trabajar, para ello voy a fijar un modelo cualquiera, en este caso regresión lineal y a partir de ahí compararé los resultados con validación cruzada.

Nota: Los tiempos de ejecución puede variar a los que aquí se presentan

Tabla 5: Comparativas en regresión lineal

Experimento	Media R^2 cv	σ cv	tiempo ajuste modelo	tiempo cv
X sin prepocesar	0.737095	0.012516	0.316216	1.886190
X solo normalizado	0.737095	0.012516	0.215509s	0.749248
X norm. sin out	0.738254	0.013038	0.293344	0.749254
X red. dim. Pearson	0.724373	0.012280	0.220314	0.575430
X PCA 72 dim	0.734363	0.013467	0.253654	0.621077
X PCA 68 dim.	0.732357	0.013831	0.264193	0.562457

De aquí se deduce que:

- Cualquier tipo de reducción de la dimensión emperora el error.
- No parece que exista ninguna relación entre la dimensión del vector de características y el tiempo que tarda en ajustar.
- La normalización influye considerablemente en el tiempo pero no en el error.
- La reducción por PCA es mejor que la nuestra hecha eliminado por el coeficiente de Pearson.

Para un estudio más detallado de los tiempos habría que repetir el experimento más veces, no lo veo del todo relevante para este caso.

Como conlusión el conjunto de datos con el que trabajaremos de aquí en adelante será el normalizado sin ouliers,el que llamamos `x_train` que ha resultado ser el que mejor resultado ha obtenido.

Transformación de los datos

Viendo la gráfica de los datos de la figura 4, podríamos pensar que una de proporcionalidad inversa podría ser una buena opción, para tenerlo en cuenta realizaremos una comparativa con dicha tranformación, una normal y otra sin con una transformación aleatoria como es la cuadrática.

Los resultados obtenidos son los siguientes:

Tabla 6: Comparativa transformaciones

Transformación	Media error cv	desv.tip cv	tiempo ajuste modelo	tiempo cv
Sin ninguna	0.738254	0.013038	0.262464s	0.728209
Inversa	0.684363	0.088954	0.689754	2.001623
Cuadrática	0.762161	0.012071	0.559633	1.556234

Conclusiones a la vista de los resultado:

- Nuestra tesis no era cierta, ya que una transformación aleatoria ha mejorado el error de la transformación inversa.
- Al aumentar la dimensión ha disminuído el error, sin embargo no tenemos ningún buen motivo para optar por ahora utilizar los datos cuadráticos, así que para evitar riesgo de overfitting seguiremos utilizando los datos de 'x_train' sin transfomar.

Planteamos regularización

Observando los coeficientes vemos que para el caso de regresión lineal tienen una media de -0.3357, desviación típica de 23.2933 y estos valores obscilan en el intervalo $[-109{,}8194, 100{,}3491]$. Luego podría ser interesante plantear regularización.

Para valores de alpha $[[0{,}0001, 0{,}01, 1, 100]$

Tabla 7: Comparativas de regularización errores

Modelo	Media R^2 validación cruzada	Desviación típica validación cruzada
Sin regularización	0.73825	0.01304
Ridge alpha = 0.0001	0.72437	0.01228
Lasso alpha = 0.0001	0.73669	0.01318
Ridge alpha = 0.01	0.73826	0.01304
Lasso alpha = 0.01	0.73531	0.01328
Ridge alpha = 1	0.73788	0.01311
Lasso alpha = 1	0.65187	0.00722

Modelo	Media R^2 validación cruzada	Desviación típica validación cruzada
Ridge alpha = 100	0.72353	0.01221
Lasso alpha = 100	-0.00041	0.00038

Observando los errores no se aprecia una mejora considerable como para asegurar que la regulación ha sido beneficiosa, de hecho, salvo para el valor Ridge alpha = 0.01 el resto de errores ha sido peor al modelo sin regularizar.

Lasso es por lo gneral peor siendo incluso extremadamente malo para alpha = 100. No pasemos por alto que a nivel teórico el error R^2 debería de ser positivo o cero, sin embargo a nivel de cálculos, el propio sklearn nos avisa que esto es posible, por el método de cálculo.

Tabla 8: Comparativas de regularización coeficientes

Modelo	Media coeficientes	Desviación típica coeficiente	Intervalo coeficientes
Sin regularización	-0.3357	23.2933	$[-109{,}8194, 100{,}3491]$
Ridge alpha = 0.0001	0.0897	11.6398	$[-36{,}2930, 26{,}0460]$
Lasso alpha = 0.0001	-0.10280	13.23585	$[-28{,}655, 33{,}710]$
Ridge alpha = 0.01	-0.33305	23.14947	$[-109{,}058, 99{,}727]$
Lasso alpha = 0.01	-0.06890	10.11854	$[-23{,}478, 26{,}716]$
Ridge alpha = 1	-0.19816	16.45576	$[-68{,}236, 66{,}293]$
Lasso alpha = 1	0.12576	1.76816	$[-4{,}889, 7{,}522]$
Ridge alpha = 100	0.09446	5.45929	$[-12{,}657, 14{,}729]$
Lasso alpha = 100	0.00000	0.00000	$[0{,}000, 0{,}000]$

En cuanto a regularización de los coeficientes podemso aprecia que para valores 0,0001 ya se nota considerablemente la mejora, aunque es notable que un mayor incremento del alpha no significa una reducción de la media, compárese por ejemplo Ridge alpha = 0.0001 y Ridge alpha = 0.01 .

Valores grandes como el caso alpha = 100 son demasiados agresivos para este problema, ya que en lasso incluso hace tender todos los coeficientes a cero.

A nivel de media y desviación de coeficientes no parece que exista una disminución considerable entre ridge y lasso.

Tabla 9: Comparativas de regularización tiempos

Modelo	Tiempo ajuste	tiempo validación cruzada
Sin regularización	0.2780	0.7299
Ridge alpha = 0.0001	0.1542	0.3140
Lasso alpha = 0.0001	7.772	14.024
Ridge alpha = 0.01	0.074	0.366
Lasso alpha = 0.01	7.468	13.940
Ridge alpha = 1	0.101	0.372
Lasso alpha = 1	0.323	0.668
Ridge alpha = 100	0.099	0.370
Lasso alpha = 100	0.068	0.236

Por lo general ridge parece tener mejores tiempos.

Conclusiones de la regularización

No parece que haya hipótesis suficientes para asegurar que la regularización mejore el error en el problema, aunque un buen motivo de selección puede ser la mejora en tiempos con el método ridge.

Selección de otros modelos y ajuste de sus hiperparámetros

Finalmente probaremos regularización por gradiente estocástico y L2 para ver si podemos mejorar

En penalizaciones usaremos l2 porque ya hemos visto que es considerablememente mejor que l1.

Para valores de alpha optaremos por los mejores según el experimento anterior `alphas = [0, 0.01, 1.0]`

Los datos generales ajustados son:

```
algoritmos = ['squared_loss', 'epsilon_insensitive']
penalizaciones = ['l2']
tasa_aprendizaje = ['optimal', 'adaptive']
alphas = [0.001, 0.0001, 1]
eta = 0.0001
```

Los datos obtenidos han sido

Tabla 10: Bondad ajuste SGD

Modelo	Media R^2 cv	Desviación típica cv
SGD 0, squared_loss, optimal, a=0.001	-379879755363378.37500	316806030106607.68750
SGD 1, squared_loss, adaptive, a=0.001	0.70043	0.01111
SGD 2, epsilon_insensitive, optimal, a=0.001	0.70078	0.01274
SGD 3, epsilon_insensitive, adaptive, a=0.001	0.42907	0.40104
SGD 4, squared_loss, optimal, a=0.0001	-872408113690554191052 8.00000	401430423639201415168 0.00000
SGD 5, squared_loss, adaptive, a=0.0001	0.70015	0.01193
SGD 6, epsilon_insensitive, optimal, a=0.0001	0.62925	0.05990
SGD 7, epsilon_insensitive, adaptive, a=0.0001	0.44876	0.37637
SGD 8, squared_loss, optimal, a=1	-725063748.16417	1449512907.71730
SGD 9, squared_loss, adaptive, a=1	0.62073	0.00773
SGD 10, epsilon_insensitive, optimal, a=1	-0.38109	0.03355
SGD 11, epsilon_insensitive, adaptive, a=1	-0.46762	0.02048

Todos son considerablemente peor.

Tabla 11: Tiempos empleados para SGD

Modelo	t.ajuste	tiempo vc
SGD 0, squared_loss, optimal, a=0.001	0.165	0.573
SGD 1, squared_loss, adaptive, a=0.001	0.220	0.587
SGD 2, epsilon_insensitive, optimal, a=0.001	0.082	0.255
SGD 3, epsilon_insensitive, adaptive, a=0.001	0.567	1.250
SGD 4, squared_loss, optimal, a=0.0001	0.104	0.352
SGD 5, squared_loss, adaptive, a=0.0001	0.234	0.566
SGD 6, epsilon_insensitive, optimal, a=0.0001	0.084	0.260
SGD 7, epsilon_insensitive, adaptive, a=0.0001	0.616	1.277
SGD 8, squared_loss, optimal, a=1	0.410	63.568
SGD 9, squared_loss, adaptive, a=1	0.194	0.527

Modelo	t.ajuste	tiempo vc
SGD 10, epsilon_insensitive, optimal, a=1	0.078	0.254
SGD 11, epsilon_insensitive, adaptive, a=1	0.257	0.529

Tabla 12: Análisis coeficientes SGD

Modelo	Media coeficientes	Desv. coef	Intervalo coeficientes
SGD 0, squared_loss, optimal, a=0.001	-16167710.23694	2528128884.563018	[−9072925316,819, 9290240617,779]
SGD 1, squared_loss, adaptive, a=0.001	0.16742	2.82429	[−7,495, 7,855]
SGD 2, epsilon_insensitive, optimal, a=0.001	0.11424	3.53939	[−9,858, 11,893]
SGD 3, epsilon_insensitive, adaptive, a=0.001	0.13799	1.30179	[−3,501, 4,082]
SGD 4, squared_loss, optimal, a=0.0001	-110164007089.55455	710643433147.06218	[−1621184110916265,495, 1498885737018,452]
SGD 5, squared_loss, adaptive, a=0.0001	0.16891	2.77359	[−7,490, 7,692]
SGD 6, epsilon_insensitive, optimal, a=0.0001	-0.06156	7.53228	[−20,290, 21,228]
SGD 7, epsilon_insensitive, adaptive, a=0.0001	0.14259	1.38753	[−3,778, 4,321]
SGD 8, squared_loss, optimal, a=1	42.91602	1267.73112	[−3008,964, 3498,335]
SGD 9, squared_loss, adaptive, a=1	0.11079	0.86456	[−2,400, 2,397]
SGD 10, epsilon_insensitive, optimal, a=1	0.01241	0.11200	[−0,170, 0,230]
SGD 11, epsilon_insensitive, adaptive, a=1	0.01202	0.10616	[−0,161, 0,219]

Conclusiones finales

A partir de los datos obtenidos en validación cruzada el modelo que creemos más oportuno es regresión lineal clásica con los datos normalizado. Vamos a proceder al cálculo de su error en test.

- El score en test de `regresion_lineal_sin_outliers_normalizados` es 0.72809

- Su error dentro de la muestra es de 0.74058

Dan resultados coerentens.

Para conseguir una aproximación final mejor entrenaré ahora con todos los datos disponibles:

Ahora el error dentro de la muestra es de 0.73810.

Y los coeficientes finales son:

```
[  -5.03464881   25.12726738  -29.63955445  -16.08989186   23.31876557
 -12.72930963    2.28827438   11.79554543    0.65189973  -11.23126874
   1.68295703   14.32638897  -23.67662571  -12.48164682   22.21200275
 -42.87020392   14.789619     21.62170753    4.63353855  -22.20847874
  -3.09023986  -10.75758111   92.41603694    4.47084164 -100.91127751
  27.27759662   17.93545442   12.91883547   -3.11178101   -8.13559299
  -8.18806856  -13.5568395    -0.59300049    4.56602126    9.14065602
   5.75605075   -6.3981041    -6.59051061   -0.11583777   10.39194084
  -2.54616392   -2.86413771   16.77665281    5.15527572  -18.37745171
   1.41183552   -6.0048152   -21.57223156   -3.79018449   26.89261847
 -11.19120282   19.97938517  -28.24282798  -15.79903521   21.55958573
  -7.49115431    9.35899058   -7.64875089    6.98899077   -4.98260296
   5.66919664   -2.16366059   23.92419635   -2.29311171  -13.02622619
   3.30095386    0.50554918  -13.88054008   -9.80820039   16.37230292
  -0.43269037  -19.63394812   29.28920034   23.77807324  -33.40295992
  29.3930802   -26.99417789    6.22070315   -0.86233777    3.2880403
 -11.42343509]
```

Matriz de confusión

A pesar de que la matriz de confusión es para datos, enteros, redondeando los datos a valores enteros podemos visualizarla para poder tener una idea de cómo se destribuyen las etiquetas y los problemas de clasificación.

Podemos ver que una gran nube de etiquetas se encuentra en la diagonal, lo que refleja que el acierto es correcto, por otra parte también se aprecia muchos para valores menores, lo cual indica que es por ello que es más difícil de clasificarlos.

Comparación bondad del ajuste con dummy approximator

Aunque la bondad del ajuste no se acerque a la perfección, si utilizáramos un aproximador dummy, su bondad sería peor, luego podemos estar satisfechos con el ajuste encontrado.

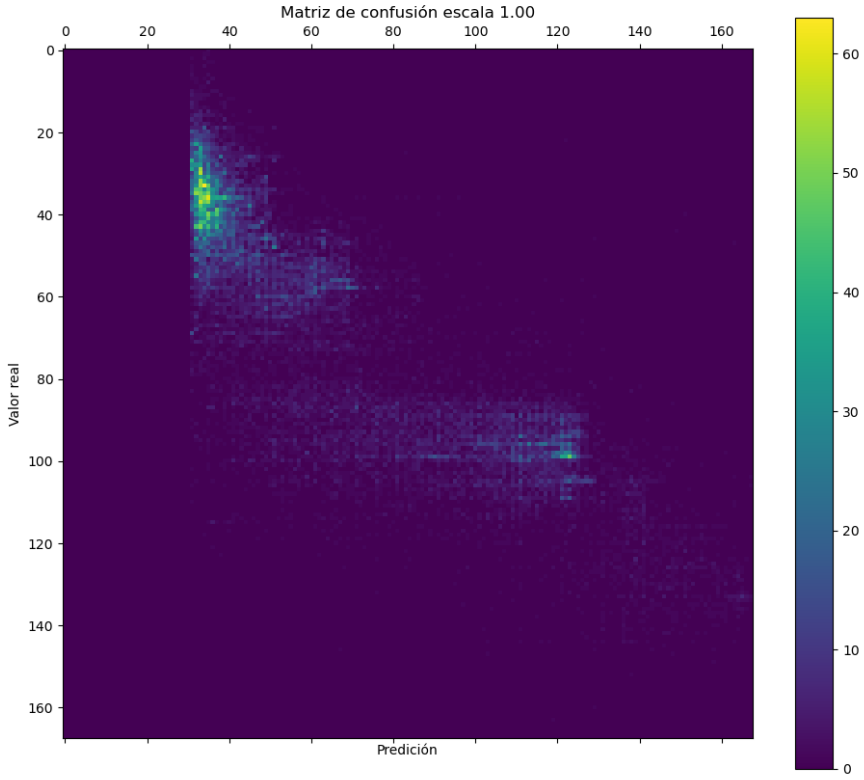


Figura 5: Matriz de confusión de todos los datos

Problema de clasificación

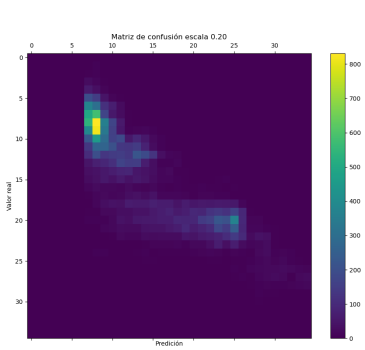
Análisis del problema

Estamos ante un problema de clasificación.

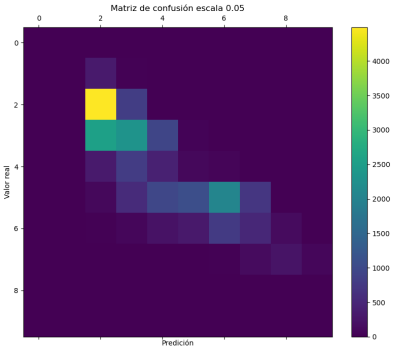
De la página web de la que se han obtenido los datos Dataset for Sensorless Drive Diagnosis Data Set

Abstract: Las características son extraídas de la corriente de un motos. El motor puede tener componentes intactos o defectuosos. Estos resultados se encuentras en 11 clases con diferentes condiciones.

Tenemos además la siguiente información:



(a) Matriz de confusión con escala 0.2 de las etiquetas



(b) Matriz de confusión con escala 0.2 de las etiquetas

- Las características del data set son multivariantes.
- Los datos son tipo números reales.
- Es una tarea de clasificación.
- En número de instancias total es 58509.
- El número de atributos es de 49.
- No faltan datos.

Descripción del tratamiento de los datos

Lectura y tratamiento inicial de los datos

El fichero tiene extensión `.txt` de texto plano, para leerlo usaremos la función escrita `LeerDatos (nombre_fichero, separador)` que tiene como pilar básico la función `read_csv` de la biblioteca de pandas.

Nota: suponemos que la estructura de carpetas es:

```
.
|- clasificacion.py
|- datos
    |-Sensorless_drive_diagnosis.txt
```

Donde `clasificacion.py` es el nombre del ejecutable de nuestra práctica, `datos` es una carpeta y `Sensorless_drive_diagnosis.txt` es el fichero que contiene los datos.

Encode

Las etiquetas ya se encuentran como enteros, luego las dejaremos de esta manera, es decir estamos utilizando una codificación por enteros. En caso de haber tenido una codificación categórica, hubiera sido interesante plantearse el one-Hot encoding.

Selección de test y entrenamiento

Comprobaremos antes si los datos están balanceados, para ello contaré el número de distintas etiquetas.

Esto lo haremos viendo el número de veces que se repite cada etiqueta, el resultado es:

Tabla 13: Comprobación del balanceo total en X.

Etiqueta	Número apariciones
1.0	5319
2.0	5319
3.0	5319
4.0	5319
5.0	5319
6.0	5319
7.0	5319
8.0	5319
9.0	5319
10.0	5319
11.0	5319

Como podemos ver está perfectamente balanceado.

Debemos determinar ahora qué datos usaremos para test y cuáles para entrenamiento.

El porcentaje que voy a usar será un 20 % de los datos reservados para test. La elección de esta se debe a heurísticas generales usadas y porque tenemos los suficientes datos para el entrenamiento.

En cuanto a las opciones de cómo separarlos estos deben ser seleccionados de manera aleatoria con una distribución uniforme. Desconozco si además el tamaño es suficientemente grande como para serpararlos directamente sin tener que ir clase por clase tomando el mismo número. Al ser homogénena, si el tamaño es lo sufiente grande es posible suponer que la selección por clases será homogénea.

Para separarlos usaré la función `sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)` de la biblioteca de scikilearn, concretamente con los siguientes parámetros:

```
ratio_test_size = 0.2
X_train, X_test, y_train, y_test = train_test_split(
    x, y,
    test_size= ratio_test_size,
    shuffle = True,
    random_state=1)
```

- `test_size` se corresponde a la proporción de los datos que usaremos para el test, está a 0.2 porque ya hemos comentado que trabajaremos con el 20 %.
- `shuffle` a `True` porque queremos coger los datos al azar.
- `random_state` es una semilla para la mezcla.

Los resultados han sido:

Tabla 14: Balanceo datos entrenamiento por clases.

Etiqueta	Número apariciones
1.0	4181
2.0	4222
3.0	4263
4.0	4254
5.0	4264
6.0	4290
7.0	4247
8.0	4275
9.0	4275
10.0	4276
11.0	4260

Vemos que la mayor diferencia es de $|4181 - 4290| = 109$ si recordamos que cada clase contaba con 5319 esto supone una diferencia de $\frac{109}{5319}100 = 2{,}0493$ es decir que en el peor de los casos estamos entrenando con dos datos más por cada cien.

Esto no me parece del todo significativo, así que continuaré sin hacerlo por clases.

Nótese que desde ahora solo trabajaremos con los datos de entrenamiento, para no cometer ningún tipo de data snooping.

Normalización

Diferencias muy grandes entre los datos podría perjudicar al modelo, luego comprobaremos antes si es necesario si es necesario normalizar los datos.

Para ello he diseñado la función `ExploracionInicial()` que muestra la media y la varianza de los datos.

```
--------------------
Resumen de las tablas
--------------------

Media
Valor mínimo de las medias -1.5019152989937367
Valor máximo de las medias 8.416765275493

Varianza
Valor mínimo de las varianzas 3.419960283480337e-09
Valor máximo de las varianzas 752.5259323408474
--------------------
```

La variabilidad entre las medias y datos es considerable, así que vamos a normalizar.

Para ello usaremos la función `class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)` ("SrandardScaler Del Paquete sklearnPreprocessing" n.d.) Según la documentación oficial a fecha de hoy, esta función normaliza las caracteríistias eliminando la media y escalando en función de la varianza, es calculado de la siguiente manera:

$$Z = \frac{X - U}{s}$$

Donde u es la media de los datos de entrenamiento o cero si el parámetro `with_mean=False` y s es la desviación típica de los datos del ejemplo y 1 en caso de que `with_std=False`.

No es más que una tipificación del estimador.

Correlación de los datos

Veamos ahora si podemos encontrar alguna relación entre las características, para ello vamos a utilizar la matriz de correlación.

No es más que una matriz cuyas entradas toman un valor entre -1 y 1. Una entrada de índice i, j representa la relación entre la característica i y j, cuando mayor sea ese valor absuloto más relacionada estará, si es negativo la relación será de proporcionalidad inversa.

Para calcularla utilizaremos `corrcoef` de la biblioteca de numpy ("CorrcoefNumpy Del Paquete Numpy" n.d.) que devuelve el el producto de los momentos de los coeficientes.

Y para visualizarla hemos utilizado la función `PlotMatrizCorrelacion`

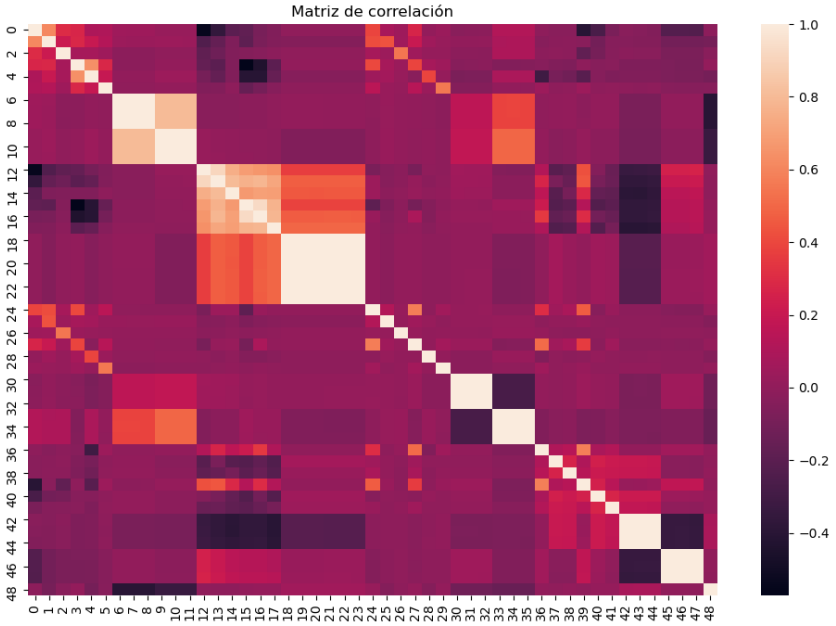


Figura 7: Matriz de correlación datos train

.

Cuando blanco sea mayor será la correlación directa, cuando más oscuro la correlación inversa.

Queda recogido el código utilizado en la función `Pearson(x,umbral,traza)`.

Aquí se muestra la correlación de las características cuyo umbrar el superior a 0.9999, en el código puede encontrar también los de `[0.9999, 0.999, 0.95, 0.9]`

Tabla 15: Coeficiente pearson para umbral 0.9999

Coeficiente	Índice 1	Índice 2
0.9999999848109186	21	22
0.9999999822104304	18	19
0.9999995890206923	9	10
0.9999995669836448	22	23
0.9999995560315011	19	20
0.9999995050949245	21	23
0.9999994842185348	18	20

Coeficiente	Índice 1	Índice 2
0.9999987036926605	6	7
0.9999940569381277	10	11
0.9999930415927681	9	11
0.9999790106652306	7	8
0.9999755087084222	6	8
0.9999725598027991	33	34
0.9999491075481474	18	23
0.9999489468171381	19	23
0.9999482678605495	18	22
0.9999480910272766	18	21
0.9999480589259921	19	22
0.9999478753629737	19	21
0.999947661434928	20	23
0.9999462814165624	20	22
0.9999460533676472	20	21
0.9999314039884454	30	31

Si además nos fijamos se cumple la propieda transitiva, esto es, si entendemos la correlación como *Si dos vectores guardan cierta correlación superior al umbral, entonces se podría decir que uno es combinación lineal del otro*

Luego podríamos aplicar la propiedad transitiva, esto es si i explica j y j explica k entonces i explica k.

Utilizaremos el critero recién explicado para reducir la dimensionalidad del vector de características, de tal manera que pueda verse como una base linealmente independiente.

Experimentamos con los umbrales 0.9999, 0.999, 0.95, 0.9 para ver cómo se reduce la dimensión.

Estas han sido las conclusiones (recordemos que el tamaño inicial del vector de características era de 49):

Tabla 16: Reducción de la dimensión por coeficiente de Pearson

umbral	tamaño tras reducción	reducción total
0.9999	38	11
0.999	34	15
0.95	32	17
0.9	30	19

Modelos a utilizar

Compararemos los modelos a través de la función de `Evaluacion`:

```
def Evaluacion( clasificador, x, y, x_test, y_test, k_folds, nombre_modelo):
    '''
    Función para automatizar el proceso de experimento:
    1. Ajustar modelo.
    2. Aplicar validación cruzada.
    3. Medir tiempo empleado en ajuste y validación cruzada.
    4. Medir la precisión.

    INPUT:
    - Clasificador: Modelo con el que buscar el clasificador
    - X datos entrenamiento.
    - Y etiquetas de los datos de entrenamiento
    - x_test, y_test
    - k-folds: número de particiones para la validación cruzada

    OUTPUT:
    clasificador
    '''
```

En ella se emplea la función `cross_val_score` ("crossvalscore Del Paquete sklearn.modelselection" n.d.).
La cabecera de dicha función es la siguiente:

```
sklearn.model_selection.cross_val_score(
    estimator,
    X, y=None, *,
    groups=None,
    scoring=None,
    cv=None,
    n_jobs=None,
    verbose=0,
    fit_params=None,
    pre_dispatch='2*n_jobs',
    error_score=nan
    )
```

Y los argumentos que nos conciernen son:

- `estimator`: el objeto usado para ajustar los datos (por ejemplo `SGDClassifier`.

- `X` array o lista con los datos a ajustar.
- `Y` array de etiquetas. (En el caso de aprendizaje automático como els el nuestro.
- `cv` Estrategia de validación cruzada, número de particiones.
- Salida: `scores` ndarray de flotantes del tamaño `len(list(cv))` que son las puncuaciones que recibe cada ejecución de la validación cruzada.

Se ha optado por esta función y no por `cross_validate` ("Cross Validate Del Paquete Sklearn.model Selection" n.d.) porque la diferencia entre estas dos funciones son que ésta segunda permite especificar múltiples métricas para la evaluación, pero éstas no nos son útiles ya que que miden cuestiones de tiempo que por ahora no nos interesa.

Como medida de la bondad del ajuste hemos usado `accuracy_score`("Sckit Learn Metrics Accuracy Score" n.d.) que devuelve un florando con la número de acierto, hemos optado por esta por tratarse de un problema de clasificación, ya que esta medida es la más intuitiva.

Constantes que vamos a utilizar para la experimentación

Además el número de épocas máximas que vamso a utilizar son 2000 (en regresión este valor se verá reducido por cuestiones de tiempo).

El número de folds que usamos es de 5, podría utilizarse cualquiera entre 5 y 10; pero por cuestiones de no prolongar inncesearíamente la ejecución del código se van a utilizar 5.

La técnica de validación usada será el cross validation

("Cross Validation, Evaluating Estimator Performance" n.d.) que además es la vista en clase de teoría.

Esta nos permite una buena idea del error fuera de la muestra sin cometer data snopping.

Modelos lineales que se van a utilizar del paquete de sklearn

SGDClassifier (**https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGD**
)

Teste estimador implementa el graciente descendente estocástico.

Vamos a normalizar los datos ya que la documentación no dice que para mejores resutlado deben de tener media cero y varianza uno.

Por defecto este ajuste soporta SVM.

Por defecto esta función utiliza la normal euclidea.

Como funciona el gradiente descendiente para clasificación en varias dimensiones
Como hemos visto en la teoría el gradiente descendente no es más que una técnica de optimización que no se corresponde a ninguna familia concreta de modelos de optimización.

Las ventajas que presenta este método son:

- Eficiencia
- Facilidad de ajuste de los datos.

Las desventajas que presenta este método son:
- Sensibla a la característica de las escalas. - Requiere parámeteros como el de regularización y el número máximo de iteraciones.

Por ser ser sensible a las escalas normalizaré los datos, además en teoría hemos visto que sí que influye el orden de los datos, luego procederé a un desordenado de los datos (con el parámetro `shuffle = True`).

Además entreré el modelo con la función de pérdida de sciki-learn llamada `hinge` ya que de esta manera será equivalente a SVM.

Otros parámetros que nos podríamos haber planteado eran:

- `loss="hinge"`: (soft-margin) linear Support Vector Machine
- `loss="modified_huber"`: smoothed hinge loss
- `loss="log"`: logistic regression

La función de pérdida hinge se define como:

$$L_{Hinge}(y, w) = max(1 - wy, 0)$$

La función `SGDClassifier` soporta clasificació combinando múltiples clasificadores binarios en un esquema OVA *one versus all*.

Esto quiere decir que para K clases el clasificador binario discrimina una clase frente a las $K - 1$ clases restantes.

Cuando llegua el momento de test, se calcula el valor de confianza, es decir cada una de las distancias con signo al hiperplano y se elege aquella que se la más salta.

https://scikit-learn.org/stable/modules/sgd.html
(después de la imagen te encontrarás la información.

("Stochastic Gradient Descent" n.d.)

Búsque del modelo

Comenzaremos con un modelos simple como es el perceptrón variando su tasa de aprendizaje entre: $[0{,}001, 0{,}01, 0{,}1, 1]$.

El resultado mejor ha obtenido una media en los coeficientes de -0.0028 .

Luego podría ser interesante directamente guiar a tal resultado usando regularización, vamos a probar con a = 0.01 y también, para comparar con un método de regresión lineal con max_iter = 20 y de regresión logística.

Tabla 17: Resultados obtenidos

Modelo	Accuray en cross-validation	Tiempo en ajuste
Regresión lineal max_iter = 20	0.9938702	2.567
Regresión logística max_iter = 20	0.99386845	2.3799
SGD Classifier, tasa variable, a = 0.01	0.8533	0.570
Perceptrón con tasa aprendizaje = 0.001	0.826	0.4367
Perceprón de tasa de aprendizaje 0.01	0.81974	0.54796
Perceptrón con tasa aprendizaje = 0.1	0.81447	0.4367
Perceptrón con tasa aprendizaje = 1	0.81178	0.85268

Podemos observar que el percentrón no mejora al primer modelo de la tabla 17 e indiferentemente de la tasa de aprendizaje su error se mantien más o menos igual. Utilizando regularización hemos obtenido un valor similar al del perceptrón, pero sin lugar a dudas los megres son los obtenidos con regresión lineal sin regularización y regresión logística, de hecho son tan buenos que no vamos a seguir explorando modelos.

Las respectivas matrices de confusión:

.

.

.

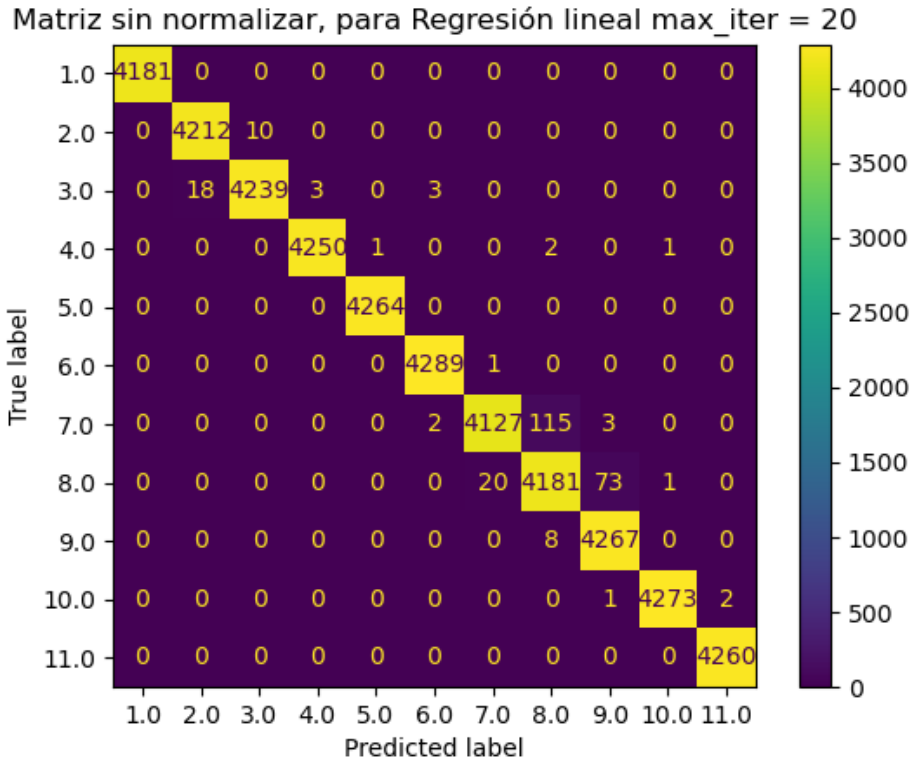


Figura 8: Regresión lineal max-iter = 20

Conclusión final

El mejor modelos seleccionado es el de regresión lineal
que tiene un error en test de 0.99307.

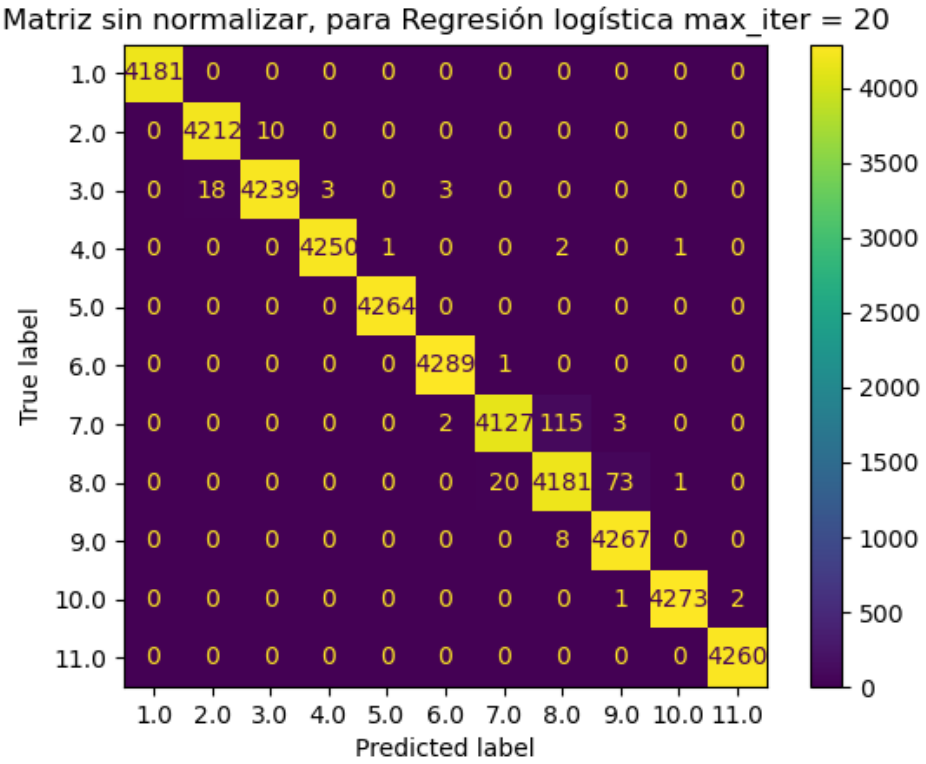


Figura 9: Regresión logística max-iter = 20

Especificaciones técnicas

Toda la experimentación se ha hecho sobre un ordenador portátil con las siguientes especificaciones

```
equipo
    description: Notebook
    product: HP Pavilion Laptop 14-bk0xx (2MF37EA#ABE)
    vendor: HP
    version: Type1ProductConfigId
    serial: 5CD7440XZ3
    width: 64 bits
    capabilities: smbios-3.0.0 dmi-3.0.0 smp vsyscall32
    configuration: administrator_password=disabled boot=normal chassis=notebook family=1

 description: CPU
         product: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
```

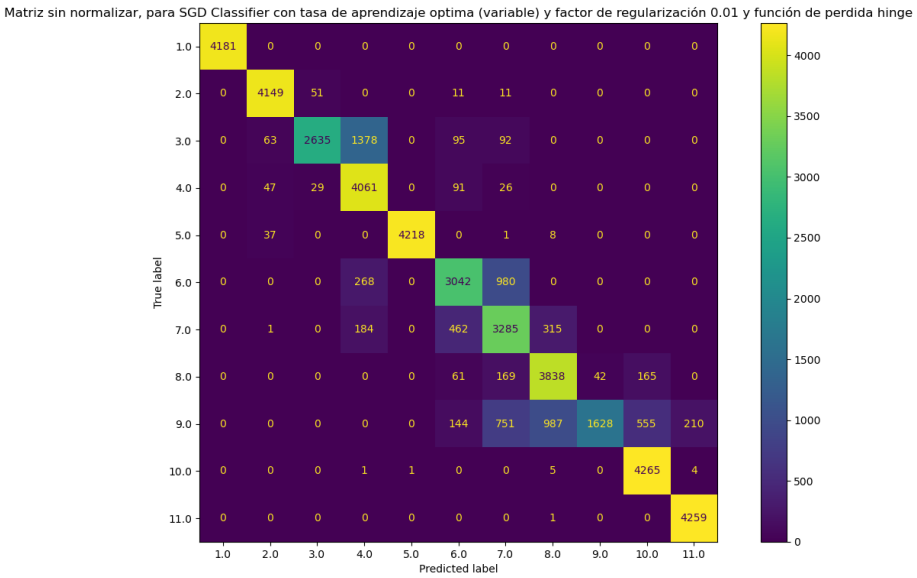
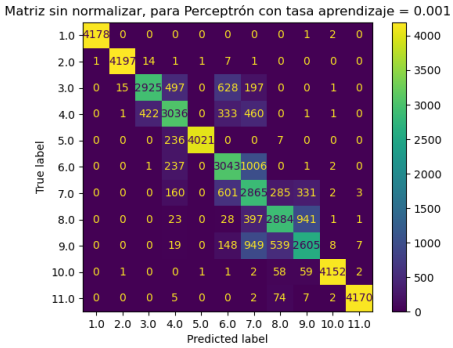
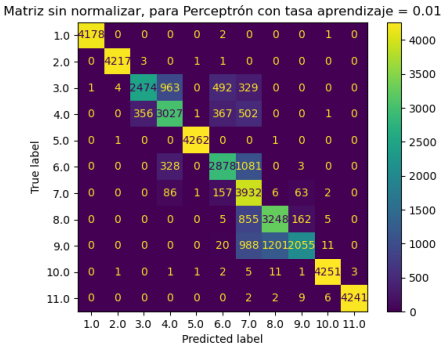


Figura 10: Evaluando SGD Classifier con tasa de aprendizaje optima (variable) y factor de regularización 0.01 y función de perdida hinge

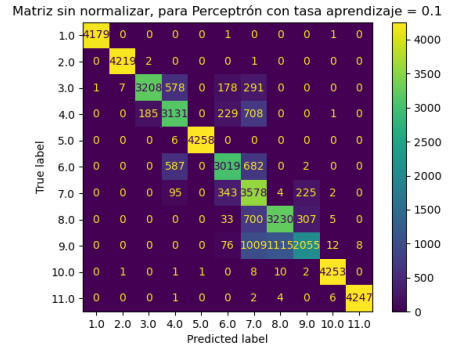
```
  vendor: Intel Corp.
  physical id: 4
  bus info: cpu@0
  version: Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz
  serial: To Be Filled By O.E.M.
  slot: U3E1
  size: 3100MHz
  capacity: 4005MHz
  width: 64 bits
  clock: 100MHz
  capabilities: lm fpu fpu_exception wp vme de pse tsc msr pae mce cx8 apic sep m
  configuration: cores=2 enabledcores=2 threads=4
*-cache:0
     description: L1 cache
     physical id: 5
     slot: L1 Cache
     size: 128KiB
     capacity: 128KiB
     capabilities: synchronous internal write-back unified
     configuration: level=1
*-cache:1
```



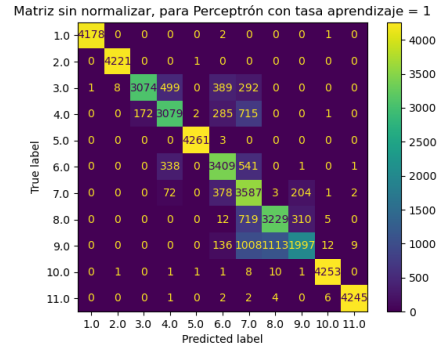
(a) Perceptrón con tasa aprendizaje = 0.001



(b) Perceptrón con tasa aprendizaje = 0.01



(c) Perceptrón con tasa aprendizaje = 0.1



(d) Perceptrón con tasa aprendizaje = 1

```
          description: L2 cache
          physical id: 6
          slot: L2 Cache
          size: 512KiB
          capacity: 512KiB
          capabilities: synchronous internal write-back unified
          configuration: level=2
     *-cache:2
          description: L3 cache
          physical id: 7
          slot: L3 Cache
          size: 3MiB
          capacity: 3MiB
          capabilities: synchronous internal write-back unified
          configuration: level=3
*-memory
     description: System Memory
     physical id: 17
     slot: System board or motherboard
```

```
      size: 8GiB
    *-bank:0
         description: SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2133 MHz (0,5
         product: M471A1K43BB1-CRC
         vendor: Samsung
         physical id: 0
         serial: 36CC9576
         slot: Bottom-Slot 1(left)
         size: 8GiB
         width: 64 bits
         clock: 2133MHz (0.5ns)
    *-bank:1
         description: SODIMM DDR Synchronous [empty]
         physical id: 1
         slot: Bottom-Slot 2(right)
*-pci
     description: Host bridge
     product: Xeon E3-1200 v6/7th Gen Core Processor Host Bridge/DRAM Registers
     vendor: Intel Corporation
     physical id: 100
     bus info: pci@0000:00:00.0
     version: 02
     width: 32 bits
     clock: 33MHz
     configuration: driver=skl_uncore
     resources: irq:0
   *-display
        description: VGA compatible controller
        product: HD Graphics 620
        vendor: Intel Corporation
        physical id: 2
        bus info: pci@0000:00:02.0
        version: 02
        width: 64 bits
        clock: 33MHz
        capabilities: pciexpress msi pm vga_controller bus_master cap_list rom
        configuration: driver=i915 latency=0
        resources: irq:129 memory:b2000000-b2ffffff memory:c0000000-cfffffff ioport
   *-generic:0
        description: Signal processing controller
        product: Xeon E3-1200 v5/E3-1500 v5/6th Gen Core Processor Thermal Subsyster
        vendor: Intel Corporation
        physical id: 4
        bus info: pci@0000:00:04.0
```

```
     version: 02
     width: 64 bits
     clock: 33MHz
     capabilities: msi pm bus_master cap_list
     configuration: driver=proc_thermal latency=0
     resources: irq:16 memory:b4220000-b4227fff
*-usb
     description: USB controller
     product: Sunrise Point-LP USB 3.0 xHCI Controller
     vendor: Intel Corporation
     physical id: 14
     bus info: pci@0000:00:14.0
     version: 21
     width: 64 bits
     clock: 33MHz
     capabilities: pm msi xhci bus_master cap_list
     configuration: driver=xhci_hcd latency=0
     resources: irq:126 memory:b4200000-b420ffff
   *-usbhost:0
        product: xHCI Host Controller
        vendor: Linux 5.10.36-2-MANJARO xhci-hcd
        physical id: 0
        bus info: usb@1
        logical name: usb1
        version: 5.10
        capabilities: usb-2.00
        configuration: driver=hub slots=12 speed=480Mbit/s
```

Recursos consultados

"CorrcoefNumpy Del Paquete Numpy." n.d. Accessed May 21, 2021. https://numpy.org/doc/stable/reference/generated/numpy.corrcoef.html.

"Cross Validate Del Paquete Sklearn.model Selection." n.d. Accessed May 25, 2021.%20%20%20%20https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_validate.html#sklearn.model_selection.cross_validate.

"Cross Validation, Evaluating Estimator Performance." n.d. Accessed May 25, 2021. https://scikit-learn.org/stable/modules/cross_validation.html.

"crossvalscore Del Paquete sklearn.modelselection." n.d. Accessed May 25, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html.

Hamidieh, Kam. 2018. "A Data-Driven Statistical Model for Predicting the Critical Temperature of a Superconductor." *Computational Materials Science* 154: 346–54. https://doi.org/https://doi.org/10.1016/j.commatsci.2018.07.052.

"Sckit Learn Metrics Accuracy Score." n.d. Accessed May 27, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html.

"Sobre normalización En Aprendizaje Automático." n.d. Accessed June 1, 2021. https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/normalize-data.

"SrandardScaler Del Paquete sklearnPreprocessing." n.d. Accessed May 21, 2021. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html.

"Stochastic Gradient Descent." n.d. Accessed May 28, 2021. https://scikit-learn.org/stable/modules/sgd.html.

"UCI Superconductivty Data Data Set." n.d. Accessed May 31, 2021. https://archive.ics.uci.edu/ml/datasets/Superconductivty+Data#.