Aprendizaje Automático - Grupo A1 (2021-2022)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 3: Ajuste de Modelos Lineales I (Regresión)

Fernando Luque de la Torre
20953877A
fl1562001@correo.ugr.es

5 de junio de 2022

Índice

1. Análisis y descripción del problema

El primer paso en la definición de un problema de aprendizaje automático es saber cuáles son cada una de las componentes del mismo. Como ya sabemos, cualquier problema de aprendizaje automático consta con 3 elementos básicos:

- Dominio: $\mathcal{X}$
- Rango: $\mathcal{Y}$
- Función objetivo: f

Antes de continuar, toda la información está extraída del repositorio de la UCI: https://archive.ics.uci.edu/ml/datasets/YearPredictionMSD [1]

1.1. ¿Qué es $\mathcal{X}$?

Como en todos los problemas de aprendizaje automático, una de las componentes es el dominio de nuestra función objetivo. En particular, para este problema una entrada de la función objetivo consta de un vector de 90 características, todas ellas números reales en las que encontramos:

- 12 características de información sobre el timbre medio de la canción
- 78 características de información sobre la covarianza del timbre de la canción

De esta forma, cada entrada para nuestra función será un vector $v \in \mathbb{R}^{90}$. Cabe destacar también que el preprocesado de estas variables no requerirá la traducción de variables binarias o categóricas. Simplemente, si queremos acelerar el proceso de aprendizaje, sería una buena idea el escalado de las variables. Esto lo analizaremos posteriormente.

1.2. ¿Qué es $\mathcal{Y}$?

En cuanto al rango de nuestra función, dado que nos encontramos ante un problema de **regresión**, la salida será un número real. Sin embargo, en nuestra muestra, la salida realmente no es un número real, sino un número natural que va desde el 1922 al 2011.

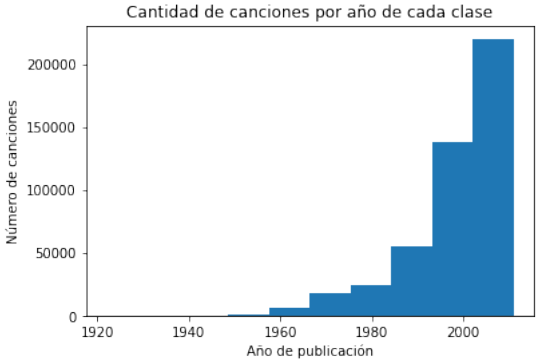
1.3. ¿Qué es f?

La función objetivo $f : \mathbb{R}^{90} \to \mathbb{R}$ será la función ideal que para cada entrada del problema $x \in \mathbb{R}^{90}$ nos da su correspondiente año de publicación. Como siempre, esta es una función desconocida y difícilmente alcanzable, por lo que el problema del aprendizaje se basará en aproximarnos lo máximo posible a esta función.

1.4. Análisis de datos

Con el objetivo de comenzar a comprender mejor el problema, una buena opción es visualizar algunos de los datos del mismo.

Años de publicación Para empezar, podemos visualizar en un histograma la cantidad de canciones que encontramos publicadas en cada año:



Como podemos ver, la distribución de canciones que encontramos a lo largo del tiempo en nuestra muestra no es regular. Se puede observar, que conforme van pasando los años, la cantidad de canciones en nuestra muestra crece enormemente. Esto se debe obviamente a la cantidad de recursos informáticos y audiovisuales, que han ido creciendo en el tiempo.

Pero, ¿cómo puede afectar esto a nuestro problema? Pues bien, el encontrar valores del rango menos representados que otros, puede provocar que las predicciones en los rangos menos representados fallen mas, ya que nuestro modelo se puede ajustar peor a ellos por ser menos representativos del total.

Además de este pequeño análisis, no me parece interesante entrar en profundidad en el análisis de ninguna de las características del problema ya que son características muy complejas las cuales llevarían un gran tiempo de estudio detrás para tratar de comprenderlas.

A pesar de esto, muestro un pequeño resumen de todas las variables, para así mostrar al menos que todas tienen variabilidad y ninguna es constante (no se incluye la captura para ahorrar espacio, se puede encontrar este resumen en el notebook).

2. Conjuntos de hipótesis

En cuanto a los conjuntos de hipótesis a utilizar utilizaremos modelos lineales, es decir, funciones de la forma:

$h(x) = \sum_{i=0}^{d} w_i x_i$

Además, el algoritmo que utilizaremos para tratar de minimizar los errores será el Desceso del Gradiente Estocástico (SGD) que nos incluye la librería scikit learn. Este algoritmo, nos permite seleccionar diferentes errores a minimizar y es por eso, que voy a estar comparando principalmente dos de ellos:

- **El error cuadrático medio**: Es el error típico en problemas de regresión. Su valor se obtiene de la suma de las distancias entre las predicciones y los valores reales al cuadrado.
- **El error Huber**: Esta es una medida de error ligeramente modificada con el objetivo de ser más consistente frente a los conocidos como *outliers*, es decir, valores atípicos de las variables. Por valores atípicos entendemos un vector de características que se asocia con una salida muy diferente a la que se asocian otros vectores con características similares. Dado la enorme cantidad de esta muestra, creo que este es un buen modelo a tener en cuenta que nos puede ahorrar llevar a cabo la eliminación de outliers en la muestra y hacer de este un proceso menos extremo.

La selección de hiperparámetros para los dos modelos será reealizada con cross validation, concretamente con la función que nos aporta sklearn: cross_val_score.

3. Definición de conjuntos de training y test

A pesar de que la documentación nos aconsejaba una partición concreta en conjuntos de train y test, en esta práctica, no vamos a utilizar esta partición sino que se realizará de forma aleatoria la siguiente división:

- Un conjunto de test que no se utilizará en absoluto, más que para estimar el E_{out}, que tendrá un tamaño del 20 % de la muestra total. Reitero que este conjunto no se tendrá en cuenta para ninguno de los cálculos (con el objetivo de evitar el data snooping).
- Un conjunto de entrenamiento (80 %) con el que: se escogerá el modelo, se ajustarán los hiperparámetros y se seleccionará la mejor hipótesis. Además, el proceso de escoger el modelo y de ajuste de los hiperparámetros conllevará una subdivisión de este conjunto de train en 5 particiones, que irán rotando entre 4 de entrenamiento y 1 de validación y que serán utilizadas para la técnica de 5-fold cross-validation.

Esta división será realizada con la función del paquete sklearn.model_selection train_test_split que nos dará la división del porcentaje de ejemplos en test y train que queramos manteniendo la proporción de cada una de las clases en los dos conjuntos.

4. Preprocesado

4.1. Escalado

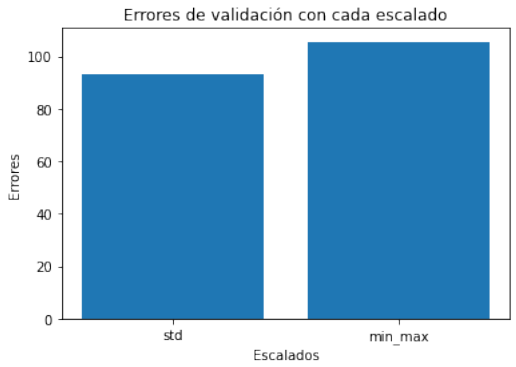
Para el escalado de los datos he considerado dos posibles alternativas:

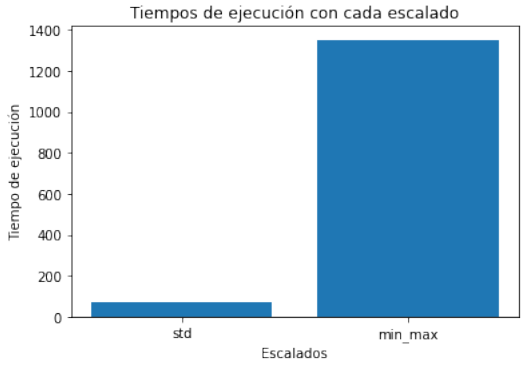
Una de las alternativas es el escalado MinMax, que nos situará todas las variables entre los valores 0 y 1 realizando la siguiente transformación:

```
X_std = (X-X.min(axis=0)) / (X.max(axis=0)-X-min(axis=0))
X_scaled = X_std * (max-min) + min
```

Por otro lado, la otra alternativa que voy a considerar es el StandardScaler, que nos transformará las variables de la siguiente forma: Sea u_j la media de un parámetro x_j y s_j su desviación típica, la transformación realizada a un parámetro x_{ij} es: $z_{ij} = \frac{x_{ij}-u_j}{s_j}$. De esta forma, lo que se consigue es hacer la media 0 y la desviación típica 1.

Con el objetivo de quedarnos con uno de estos escalados llevamos a cabo un pequeño experimento:



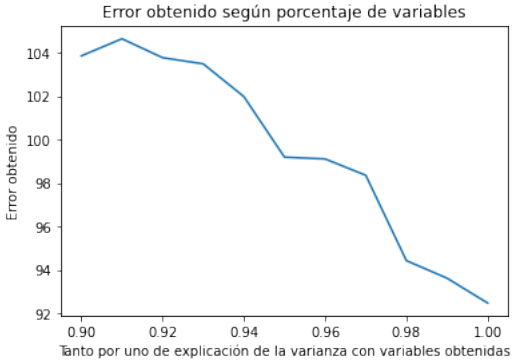


Como podemos observar, los resultados son claramente concluyentes. Para un mismo modelo (que he escogido arbitrariamente el SGDRegressor con error cuadrático medio, aunque esto no quiere decir que este sea el modelo que finalmente elegiré), además de que conseguimos obtener un error cuadrático inferior, los tiempos de ejecución del algoritmo de ajuste con el escalado estándar son infinitamente menores que con el escalado MinMax. Es por eso que a partir de aquí, se aplicará el escalado estándar a las variables.

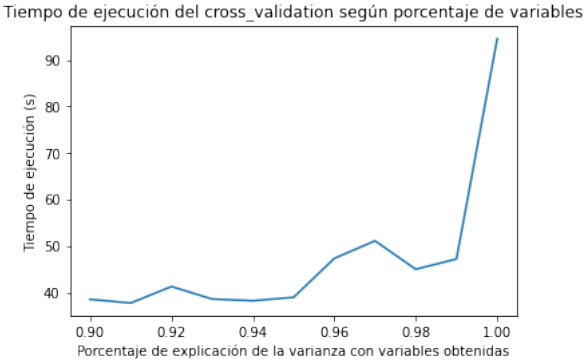
4.2. Reducción de dimensionalidad

La reducción de dimensionalidad se basa en eliminar características en función de la importancia que tienen en la distribución general. En el PCA (*Principal Component Analysis*), normalmente se ordenan las características en función de cuánto aporten a la varianza de la distribución principal y tras esto, nos quedamos con las variables que expliquen un cierto porcentaje de la varianza.

Con el siguiente experimento



Podemos observar que la cantidad de variables con las que nos quedamos tiene una relación más o menos directa con la calidad de nuestro ajuste. Sin embargo, como podemos observar aquí:



en lo que a tiempos de ejecución se refiere, el reducir en tan solo un 2 % la explicación de la varianza nos supone una mejora enorme en tiempo. Es por eso que, dado que me facilita el trabajo, nos quedaremos con un 98 % de explicación de la varianza para el PCA. Destacar que esto supone la eliminación de 13 variables, quedándonos así con 77 variables.

5. Métrica de error

En cuanto a la métrica de error, dado que estamos en un problema de regresión, la métrica utilizada será el error cuadrático medio (mse).

Cabe diferenciar que el mse será la **métrica de error**, sin embargo, tal y como he comentado antes, la función de pérdida variará según los dos modelos que vamos a comparar.

En el caso del modelo que use el mse también como función de pérdida, coincidirá con la métrica de error.

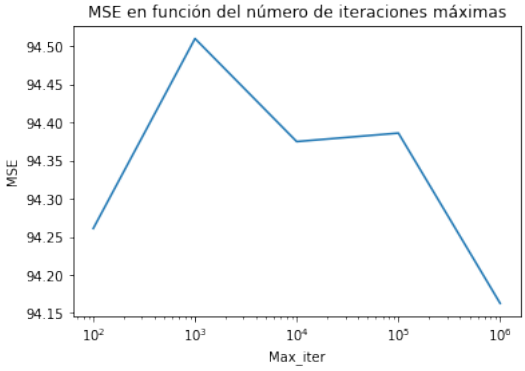
Por tanto, para el otro modelo que utiliza como función de pérdida el error Huber, también utilizaremos como métrica de error el mse. Como pequeña aclaración, el error Huber aplica el error cuadrático medio a las muestras que cumplan que: $|y - Xw|/\sigma < \epsilon$ y el error absoluto a las que no lo cumplan. Esto hace que el modelo sea más robusto frente a los *outliers*. En este modelo ϵ es un hiperparámetro que posteriormente ajustaremos y σ es un parámetro que se ajustará en el entrenamiento.

6. Parámetros y regularización

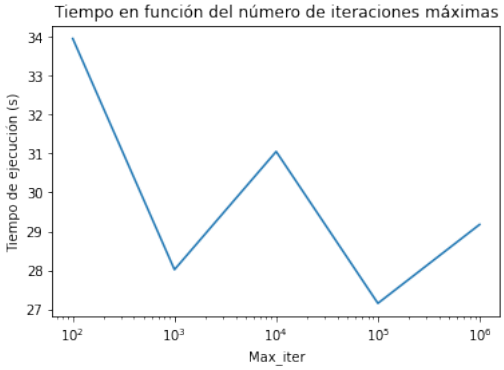
En esta sección nos encargaremos de escoger el valor de los diferentes hiperparámetros de nuestro modelo. Para ello, utilizaremos 5-fold cross_validation sobre los diferentes modelos que surgen de combinar diferentes valores de hiperparámetros. Esto nos dará tantos Errores de Validación medios como modelos propongamos y nos quedaremos con el modelo que mejor resultado de (menor error). Este proceso será realizado automáticamente por la clase GridSearchCV de sklearn la cual se encargará de hacer todas las combinaciones de los parámetros que le digamos y de quedarse con la mejor opción.

6.1. Número máximo de épocas

Sin embargo, antes de empezar a realizar los experimentos con cross validation, voy a tratar de reducir al máximo el número máximo de épocas con el objetivo de optimizar los tiempos e ejecución y que los diferentes entrenamientos (que serán bastantes dadas las diferentes combinaciones) tarden lo menos posible. Para ello, ejecuto 5-fold cross validation (con la función de sklearn cross_validation_score, que ya realiza las particiones y nos dice los aciertos en cada iteración) cambiando únicamente el valor del hiperparámetro max_iter y con los valores recomendados por sklearn del resto de hiperparámetros, obteniendo los siguientes resultados:



Como vemos, el número máximo de iteraciones apenas tiene influencia en el error cuadrático obtenido.



Además vemos que el tiempo de ejecución no es directamente dependiente del parámetro max_iter ya que el algoritmo, además de este tiene otros criterios de parada. Por eso, aunque en este experimento en concreto no haya sido el mejor tiempo, nos quedamos con un valor de épocas máximas de 100 para que, en el caso de que el algoritmo no pare por otros criterios, lo haga por este sin tardar demasiado.

6.2. Función de pérdida, tipo de regularización, λ y tipo de learning rate

Tal y como comente previamente, ahora vamos a testear con 5-fold cross validation los diferentes tipos de combinaciones de estos parámetros.

Para ello, he escogido las siguientes alternativas:

- Tipo de error: como he comentado previamente, el mse y el error huber
- Regularización: la regularización L1 y la L2
- λ: {0.001, 0.01, 0.1}

- Tipo de learning rate: entre optimal e invscaling. Ambos learning rates que se van adaptando conforme el algoritmo evoluciona

Ahora, ejecutamos el experimento GridSearchCV y tras una larga espera (en concreto en collab ha tardado en torno a 30 minutos), obtenemos el modelo que optimiza el error de validación medio.

```
[ ] print(pla.best_params_)

    {'alpha': 0.01, 'learning_rate': 'optimal', 'loss': 'squared_error', 'penalty': 'l1'}

[ ] modelo = SGDRegressor(penalty='l1',alpha=0.01, max_iter=100,learning_rate='optimal')
    modelo.fit(X,y)

    SGDRegressor(alpha=0.01, learning_rate='optimal', max_iter=100, penalty='l1')
```

7. Selección de hipótesis

Una vez escogido el modelo, procedemos a llevar a cabo el ajuste (entrenamiento):

```
[26] from sklearn.linear_model import SGDRegressor

     modelo = SGDRegressor(penalty='l1',alpha=0.01,max_iter=100,learning_rate='optimal')
     modelo.fit(X,y)

     SGDRegressor(alpha=0.01, learning_rate='optimal', max_iter=100, penalty='l1')

     from sklearn.metrics import mean_squared_error

     print("E_in: ",mean_squared_error(y, modelo.predict(X)))

     E_in:  93.53388031502975
```

7.1. Comparación estimador naive

Con el objetivo de tener una referencia con la que comparar nuestro estimador, vamos a crear un estimador naive. En este caso, el estimador que utilizaré será la media. Es decir, dado un conjunto de entrenamiento, se calculará la media de las salidas del mismo y ese será el valor que el modelo prediga para todas las entradas (ignora los parámetros).

Para ello, voy a utilizar la clase DummyRegressor del paquete sklearn.dummy, a la que con el parámetro strategy=’mean’ le indico que utilice la media como predictor. Por tanto, empleamos ahora 5-fold cross validation con ambos predictores y comparamos los resultados:

```
print("Relación desempeño SGD vs Media: ", -Evs['SGD'].mean(),", ", -Evs['avg'].mean())

Relación desempeño SGD vs Media:  93.57876489632613 ,  120.00560257326154
```

Como podemos observar, aunque los resultados mejoran ligeramente, tampoco obtenemos una mejora drástica con respecto al estimador naive. Esto se puede deber principalmente a que la complejidad de nuestro modelo no es suficiente para este problema y quizás se debería abordar con otros modelos no lineales como redes neuronales.

7.2. Otra métrica de error

Además de la medida que nos aporta el mse, yo personalmente, he decidido, con el objetivo de analizar el hecho de que el modelo se ajuste peor en los primeros años de la muestra dado que hay menos canciones que en los últimos, he creado la siguiente métrica:

He dividido la muestra en tramos de 20 años, desde el 1920-1940 hasta el 2000-2020 y hago lo siguiente: Para cada elemento de la muestra, calculo el cuadrado de la distancia entre el valor real y el valor predicho y lo sumo al total de su tramo; además incremento en 1 el contador de los elementos de su tramo. De esta forma, al finalizar, divido las sumas de los cuadrados de las distancias de cada tramo entre cada contador y obtengo los siguientes valores:

```
[ ]  print(sq_error_divided(y,modelo.predict(X)))

     {1920: 3236.867911029152, 1940: 1392.9339831051866, 1960: 523.595689456425, 1980: 45.74294887624345, 2000: 48.42445427530583}
```

Como vemos, el error cuadrático en los tramos menos representados es mucho mayor. Esto se debe a que nuestro modelo está intentando ajustar mejor los individuos que están más fuertemente representados, es decir, los que pertenecen a años posteriores y "se olvida"de estos elementos de los primeros años que suponen un mínimo porcentaje de la muestra total.

De nuevo, quizás esto sea problema de la poca complejidad de nuestro modelo y sería una idea fundamental tratar de resolverlo con métodos no lineales, pero dada la limitación de tiempo de esta práctica, no lo haré.

8. Acotación E_{out}

Como vimos en la práctica anterior, hay dos modos de acotar el valor de E_{out}. La primera de ellas es con el valor de E_{in}:

$$E_{out} \leq E_{in}(h) + \sqrt{\frac{8}{N} \log \frac{4((2N)^{d_{vc}}+1)}{\delta}}$$

Sin embargo, en esta cota, el término $(2N)^{d_{vc}}$ nos penalizaba mucho, haciendo que esta fuera demasiado laxa y no nos aportara información.

Por otro lado, la otra cota surgía de la idea de utilizar un subconjunto de test que se utilizara únicamente para evaluar la calidad del ajuste fuera de la muestra.

Este subconjunto de test es el subconjunto extraído al inicio de la práctica. Es importante que este subconjunto sea utilizado únicamente para esta estimación de la cota ya que si no, estaríamos haciendo *data snooping*. Además, dado que el conjunto de test se guardó tal y como se extrajo, debemos aplicar a las características **el mismo preprocesado** que aplicamos al conjunto de entrenamiento, esto es, normalizar los datos con el escalado estándar y aplicar la misma reducción de características que aplicamos al conjunto de entrenamiento.

Una vez aplicados estos mismos preprocesados, procedemos a calcular la cota de la siguiente manera:
$E_{out} \leq E_{test}(h) + \sqrt{\frac{1}{2N} \log \frac{2|H|}{\delta}}$

En esta fórmula, ya que el conjunto de test no se ha usado en ningún momento del entrenamiento, el término $|H| = 1$, ya que solo tenemos una función en la clase, la función obtenida del entrenamiento del modelo. Por tanto con un grado de confianza del 95 %, el error E_{out} será:

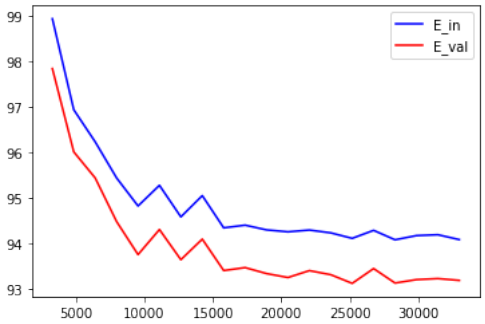
```
print("Cota Eout: ", mean_squared_error(y_test, modelo.predict(X_test))+np.sqrt(1/(2*y_test.size)*np.log(2/0.05)))
```

```
Cota Eout:  91.2247133504684
```

Como vemos, el haber conseguido extraer un conjunto de test muy grande, nos ha permitido reducir en gran cantidad el segundo término de la cota y hacer que esta sea bastante restringida.

9. Curvas de aprendizaje

Vamos a analizar ahora el comportamiento del modelo con diferentes tamaños del conjunto de entrenamiento. El procedimiento es el indicado en la presentación de la práctica, obteniendo los siguientes resultados:



Como podemos observar, la curva de aprendizaje presenta una forma poco común ya que el error de validación (con el que no se entrena) está por debajo incluso del error dentro de la muestra. Sin embargo, esto no es tan extraño e incluso es una buena señal si lo

unimos al hecho de que las dos curvas se encuentran muy juntas, ya que nos indica que nuestro modelo no está sobreentrenando en absoluto y es capaz de llevar a cabo una buena generalización. Esto es fruto del buen ajuste de los hiperparámetros de regularización.

10. Caso práctico

Suponga ahora que Ud. debe realizar este ajuste para una empresa que le ha proporcionado los datos, sin distinción entre training y test. ¿Cuál sería el mejor modelo que les propondría, y qué error E_{out} les diría que tiene? Justifique todas las decisiones Pues bien, en el caso de que la empresa nos aportara un único conjunto de datos, se podrían adoptar varios enfoques distintos, en función del tamaño de este:

- En el caso de que hubiera elementos de sobra, un posible enfoque sería el adoptado en esta práctica: en primer lugar separar una parte como conjunto de test y olvidarnos de este hasta el momento en que nuestro modelo esté entrenado y utilizarlo **únicamente** para estimar el valor de E_{out} y no para nada más. El proceso de selección de hiperparámetros lo realizaría con algún tipo de cross-validation y el entrenamiento del modelo con los parámetros escogidos con el conjunto de train en su totalidad.

- En el caso de que tengamos bastantes, pero no suficientes elementos, una buena idea sería omitir la separación del conjunto de train. Es decir, con todo el conjunto, realizaríamos la técnica de cross-validation para elegir hiperparámetros, pero ahora también para elegir los parámetros (pesos) quedándonos con el modelo que menor E_{cv} nos diera. Además, con este E_{cv} concreto, podríamos establecer también una cota para E_{out} que quizás es más laxa que la vista en el punto anterior, pero al fin y al cabo, nos permite trabajar con una menor cantidad de datos.

- Por último, en problemas en los que dispongamos de muy poca cantidad de datos, la técnica que utilizaría sería la de *leave-one-out*, es decir, si N es el tamaño de la muestra que nos aportan, entrenar N modelos dejando como validación un único elemento fuera del conjunto de entrenamiento. De nuevo, nos quedaríamos con el mejor de los modelos tanto en hiperparámetros como en parámetros y podríamos establecer una cota del error fuera de la muestra con el error obtenido en el elemento que hemos dejado fuera.

Como ya he dicho, en función del tamaño del conjunto de los datos, propondría del primer punto al último en orden de preferencia, teniendo en cuenta cuál es el que nos permite utilizar ese tamaño en concreto.

Por último, aclarar que sería una alternativa muy mala validar y entrenar con el mismo conjunto de datos. ¿Por qué? Pues bien, como nos decía la desigualdad de Hoeffding:

$P[|E_{in} - E_{out}| > \epsilon] \leq 2Me^{-2\epsilon^2 N}$

Donde M es el tamaño de la clase de funciones de la que elegimos nuestra función. Entonces, al no hacer distinción entre conjunto de training y test, el tamaño de la clase de funciones en esta expresión sería infinito (realmente sería finito dado que los ordenadores no pueden representar todos los reales, pero igualmente, muy grande) y por tanto, no se podría establecer una cota lo suficientemente pequeña como para asegurar un buen rendimiento de nuestro modelo fuera de la muestra.

11. Bibliografía

Referencias

[1] Dheeru Dua y Casey Graff. *UCI Machine Learning Repository*. 2017. URL: `http://archive.ics.uci.edu/ml`.