Trabajo Fin de Máster

Máster En Investigación En Inteligencia Artificial

Curso 2020/2021

Curvas de Aprendizaje en la Optimización Bayesiana de Hiperparámetros

---

Autor:
OSKITZ RUIZ SARRIAS

Tutores:
FELIX HERNANDEZ DEL OLMO
ELENA GAUDIOSO VAZQUEZ

Resumen

La optimización bayesiana (OB) es uno de los métodos más eficientes para la optimización de hiperparámetros (H) en los algoritmos de aprendizaje automático. Para evaluar el rendimiento de cada configuración de H, la OB utiliza la validación cruzada; lo que en entornos de grandes bases de datos supone un alto coste computacional. Para acelerar la OB en la literatura se han propuesto cuatro estrategias principales: (1) eliminación de combinaciones poco prometedoras, (2) terminación anticipada de evaluaciones innecesarias, (3) paralelización de los cálculos en diferentes unidades, (4) hacer uso de funciones de adquisición más eficientes.

Las curvas de aprendizaje (CA) son funciones que enfrentan el número de datos con el que es entrenado un modelo al rendimiento que obtiene tras ser entrenado con dicha cantidad de datos. Poder predecir, partiendo de los puntos iniciales de la CA, el recorrido de la misma, permite estimar el rendimiento del modelo sin necesidad de entrenarlo con todos los datos disponibles. Por lo tanto, el uso de la predicción del rendimiento mediante CA, permitiría reducir el coste computacional de la OB al acelerar las evaluaciones de rendimiento. Y esto último es lo que intenta demostrar este trabajo.

En la presente investigación se ha trabajado con problemas de clasificación binaria. Para ello, se han utilizado dos algoritmos de aprendizaje automático: el *Support Vector Machine* (SVM) y una red neuronal (RN) del tipo perceptrón multicapa. El objetivo final de la fase experimental de este trabajo, ha sido comparar el coste computacional requerido al realizar la OB midiendo el rendimiento con CA o con la validación cruzada. Con esto, se han obtenido resultados que muestran una reducción significativa en el tiempo de cómputo de la OB al usar la predicción por CA.

Palabras clave: Optimización de Hiperparámetros, Optimización Bayesiana, Curvas de Aprendizaje.

Índice

1. Introducción

Normalmente, los algoritmos de aprendizaje automático se componen, entre otros, de hiperparámetros que determinan cómo el algoritmo, partiendo de los datos, ajusta el modelo en la fase de entrenamiento. Estos valores son establecidos antes del proceso de aprendizaje. Por lo tanto, uno de los objetivos en la creación de estos modelos es encontrar, en un período de tiempo razonable, la configuración de hiperparámetros que obtenga el mejor rendimiento posible. Este proceso se denomina optimización o ajuste de hiperparámetros, y es vital para obtener resultados de calidad. Desafortunadamente, en la práctica no siempre es posible prever como influirá la variación de un hiperparámetro concreto en la efectividad del algoritmo. Por lo tanto, cómo optimizar los hiperparámetros se convierte en un tema clave en la creación de un modelo predictivo [1]. En numerosas ocasiones, la selección correcta de estos valores puede marcar la diferencia entre un buen modelo y uno mediocre.

Se pueden diferenciar dos tipos de métodos de optimización de hiperparámetros: la búsqueda manual y los métodos de búsqueda automática [1]. La búsqueda manual, como su nombre indica, consiste en la búsqueda por prueba y error, y depende de la intuición fundamental y la experiencia del usuario. Evidentemente, la búsqueda manual se puede convertir en un proceso largo y tedioso en aquellas personas sin experiencia. Y además, los resultados no suelen ser replicables, es decir, lo que haya servido en un problema no tiene por que funcionar en otro. A su vez, en entornos de múltiples hiperparámetros, este proceso se complica aún más, dado que el espacio de búsqueda de estas configuraciones es muy grande; además, para cada hiperparámetro, puede haber un número infinito de valores posibles (este es el caso de cualquier hiperparámetro continuo).

Teniendo en cuenta lo anterior, parece sensato explorar diferentes configuraciones de alguna otra manera que no sea por selección manual. Y esto es precisamente lo que buscan los métodos de búsqueda automática. Si la retroalimentación del rendimiento se diera de inmediato, podría aplicarse la fuerza bruta para buscar la configuración óptima. Pero entrenar y evaluar un modelo con cada configuración suele ser costoso, por lo que la fuerza bruta no es una estrategia útil. Los métodos de búsqueda automática más utilizados, acotan el espacio de búsqueda con el objetivo de reducir el coste computacio-

nal del proceso. Por ejemplo, el principio de la búsqueda en cuadrícula realiza una búsqueda exhaustiva sobre un conjunto de valores predefinidos para cada hiperparámetro, donde se entrena un modelo de aprendizaje automático con cada combinación de posibles valores. Aunque este método teóricamente puede obtener el óptimo global, sufre la maldición de la dimensionalidad, es decir, la eficiencia del algoritmo disminuye rápidamente a medida que aumenta el número de hiperparámetros que se ajustan y aumenta el rango de valores de los hiperparámetros.

Para superar ese inconveniente y reducir aún más el espacio de búsqueda se han propuesto otras alternativas, como la búsqueda aleatoria [2] o la optimización bayesiana [3]. Siendo la segunda de estas la que ha demostrado superar a otros algoritmos de optimización globales en una serie de funciones de referencia de optimización [4].

El objetivo de la presente investigación, se centra en reducir el coste computacional de la optimización bayesiana haciendo uso de las curvas de aprendizaje. Por lo que se pretende aportar un mecanismo para convertir más eficiente uno de los algoritmos de optimización de hiperparámetros más utilizado. La optimización bayesiana hace uso de la validación cruzada para medir el rendimiento de una configuración determinada de hiperparámetros de un algoritmo, siendo este procedimiento extremadamente costoso, en términos computacionales, al trabajar con grandes bases de datos. La hipótesis que intenta verificar esta investigación es que, sustituyendo la estimación de rendimiento con validación cruzada por la estimación mediante curvas de aprendizaje se podría acelerar la optimización bayesiana al no ser necesario entrenar cada configuración con gran parte de los datos disponibles. La consecuencia directa de esto, es que se estaría haciendo un avance en la eficiencia de la optimización de hiperparámetros en entornos *Big Data*.

La metodología seguida en el trabajo para poder aceptar o refutar la hipótesis principal, se ha dividido en tres partes. En primer lugar, se ha realizado un estudio del estado del arte de las diversas áreas involucradas en el trabajo: curvas de aprendizaje, optimización de hiperparámetros y mejora en la eficiencia de la optimización bayesiana. En segundo lugar, se han estudiado las bases teóricas de los métodos implementados en la investigación: curvas de aprendizaje y optimización bayesiana. Por último, en la fase experimental, se ha estudiado como implementar el método propuesto y se ha procedido

a verificar la hipótesis planteada. En términos generales, se ha seguido una metodología clásica en este tipo de investigaciones con el objetivo de que el procedimiento seguido sea lo más claro posible.

La presente memoria está estructurada de la siguiente manera: en el segundo capítulo, se analiza el estado del arte de las áreas involucradas en el trabajo. En el tercer capítulo, se estudian los métodos utilizados en el trabajo, comenzado por analizar la manera en que se puede predecir las curvas de aprendizaje, y los algoritmos diseñados para ello, seguido por la optimización bayesiana y finalizando con el método propuesto. El cuarto capítulo se centra en desarrollar la fase experimental del trabajo. Por último, el quinto capítulo expone las conclusiones obtenidas en esta investigación, los futuros trabajos y los aspectos éticos relacionados.

2. Estado del arte

En este segundo capítulo se estudia el estado del arte de los temas trabajados en la investigación. Es por ello que este capítulo se divide en cuatro apartados. En los dos primeros, se estudia, respectivamente, el estado del arte de las curvas de aprendizaje y de la optimización de hiperparámetros. Posteriormente, se analizan los trabajos relacionados con la mejora de la optimización bayesiana. Y por último, partiendo del estado de la cuestión planteado, se exponen las conclusiones obtenidas.

2.1. Curvas de aprendizaje

Los avances hechos en las últimas décadas, han posibilitado que la generación, el almacenamiento y el acceso a los datos hayan aumentado. A su vez, el desarrollo de técnicas de inteligencia artificial han hecho posible procesar y extraer información útil de estos datos. En el campo del aprendizaje supervisado, para generar modelos predictivos, los métodos basados en esta técnica necesitan una base de datos de entrenamiento anotada; es decir, una muestra donde haya datos con información tanto de la variable que se quiere predecir (*variable dependiente*), como de las variables en las que se basarán para hacer las predicciones (*variables independientes*).

La lógica dice que, con cuanta mayor cantidad y calidad de datos se entrenen estos modelos, mejor rendimiento obtendrán. Pero tanto por la falta de datos anotados en algunos casos [5], como por el tiempo de cómputo necesario en otros, en muchas ocasiones no será posible entrenar los modelos con tamaños de muestra tan grandes como se quiera. Además, la ley de los rendimientos decrecientes dice que, los modelos exhiben un comportamiento en el que, en algún momento agregar más datos solo aumenta marginalmente su rendimiento [5] [6]. Esto puede ser visualizado utilizando las curvas de aprendizaje. La curva de aprendizaje, se grafica enfrentando el rendimiento de un modelo al tamaño muestral con el que ha sido entrenado. Con esto se ve que en la mayoría de los casos, las curvas muestran un rápido aumento de rendimiento, seguido de un pequeño aumento de rendimiento y, finalmente, una convergencia con un aumento de rendimiento muy marginal [5]. En la Figura 1, tomada de [5], se puede ver una curva de aprendizaje genérica de un modelo de clasificación.

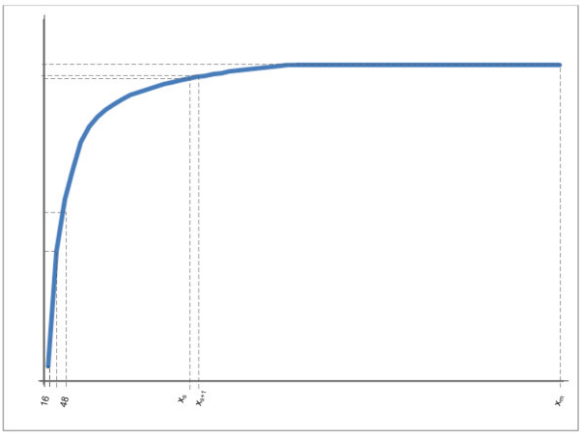


Figura 1: Curva de aprendizaje genérica [5].

Por lo que ante esta situación surge la siguiente pregunta: ¿cuál debería ser el tamaño del subconjunto de entrenamiento para alcanzar un determinado rendimiento? [5]. Para responder a esta pregunta, muchas investigaciones se han dedicado a estudiar e intentar predecir la curva de aprendizaje partiendo de la disposición inicial de ésta, puesto que esto permitiría estimar el rendimiento asociado al tamaño muestral de entrenamiento.

Se ha visto que la aplicación de este método puede aportar diversos beneficios en distintas áreas de trabajo. Por mencionar dos ejemplos:

- En entornos de trabajo en los que la obtención de datos anotados supone un coste económico o temporal, este método permite poder precisar el tamaño muestral necesario para conseguir un determinado rendimiento con el modelo; reduciendo así posibles sobrecostos o retrasos innecesarios [5] [7].
- En entornos de trabajo de Big Data, permite poder detener el entrenamiento de un modelo cuando se llegue a la meseta del aprendizaje; reduciendo así los costes computacionales [7].

Entrando en materia, se observa que las curvas de aprendizaje están asociadas con varias áreas de investigación, como la determinación del tamaño de la muestra [8], el aprendizaje activo [9] y el muestreo progresivo [10].

La determinación del tamaño muestral es el área de estudio que se encarga de determinar el tamaño de muestra necesario en el diseño de un estudio. Está muy unida a la estadística, con aplicación directa en áreas como la investigación sanitaria [8], donde es muy costoso reclutar nuevos pacientes. Debido a dicho coste, el objetivo suele ser definir el tamaño de la muestra mínimo para lograr suficiente poder estadístico como para probar o refutar una hipótesis. En el campo del aprendizaje automático, el objetivo de determinación del tamaño muestral suele ser encontrar la cantidad de instancias necesarias para alcanzar un rendimiento determinado en un modelo. El aprendizaje activo y el muestreo progresivo, por su parte, son ramas de investigación que utilizan tamaños crecientes de datos para generar modelos. En ambos casos, se comienza con una cantidad muy pequeña de datos de entrenamiento, y se añaden nuevos hasta que se cumplen unos criterios de terminación preestablecidos. En el caso del aprendizaje activo, se intenta seleccionar iterativamente las instancias más informativas para el modelo [9]. En cambio, el muestreo progresivo se centra más en minimizar la cantidad de instancias de entrenamiento, logrando el mejor rendimiento posible [10]. La diferencia entre el muestreo progresivo y la determinación del tamaño muestral es que, el muestreo progresivo asume que hay un número ilimitado de datos y no predice el tamaño muestral necesario para alcanzar un objetivo de rendimiento concreto [5].

Los estudios realizados por de Mukherjee y col. [6] en 2003 y Figuero y col. [5] en 2012 son el punto de partida de muchas otras investigaciones que han estudiado el uso de las curvas de aprendizaje en modelos de aprendizaje supervisado. Mukherjee y col. estudiaron como ajustar las leyes de potencia inversa a curvas de aprendizaje logradas empíricamente para pronosticar el rendimiento de los modelos con tamaños de muestra más grandes. Este enfoque se basa en los resultados que demostraban que las curvas de aprendizaje generalmente siguen la ley de potencia inversa [11]. Según esta ley, el *error* y el *Accuracy*, respectivamente, quedarían así representados en función del tamaño muestral con el que un modelo haya sido entrenado [5]:

$$e(n) = a + b \cdot n^c, \tag{1}$$

$$A(n) = (1 - a) - b \cdot n^c, \tag{2}$$

donde:
$0 < E[\ a\ ] \ll 1$ y $E[\ c\ ] \in [-1, 0]$ siendo,
$a \equiv$ *error mínimo alcanzable;* $b \equiv$ *tasa de aprendizaje;* $c \equiv$ *tasa de caída.* [5]

Mukherjee y col. introdujeron la idea de ajustar las curvas de aprendizaje mediante la ley de potencia inversa utilizando optimización de mínimos cuadrados no lineales. El método se probó con varios conjuntos de datos iniciales relativamente pequeños ($n \in [53, 280]$), obteniendo diferencias entre los errores de clasificación previstos y reales de entre un 1 % y un 7 %. Figueroa y col. por su parte, tomando como referencia el trabajo anterior, propusieron un método alternativo para ajustar las curvas de aprendizaje en el que se ponderan los puntos empíricamente obtenidos, dando más importancia a los que se han calculado con un mayor conjunto de entrenamiento. Evaluando su método demostraron que las predicciones eran mucho más precisas que las obtenidas por Mukherjee y col.

Tras los aportes de los estudios mencionados anteriormente, han sido muchas las investigaciones que se han desarrollado con nuevas contribuciones. Una de las áreas donde se ha centrado la investigación, ha sido en la aplicación de las técnicas de predicción de la curva de aprendizaje a entornos de *Big Data*. En este caso, se puede destacar el trabajo [7] de A.N.Richter y T.M.Khoshgoftaa. En este estudio, realizan una aproximación del método propuesto por Figueroa y col. en tres grandes conjuntos de datos desequilibrados. Concluyen que el método es efectivo para conjuntos de datos con millones de instancias. Y finalmente, plantean como futuros trabajos evaluar más clasificadores y conjuntos de datos con este método de estimación, creando nuevos modelos matemáticos o estadísticos que pueden realizar una determinación del tamaño de la muestra aún mejor.

Las dudas que surgen al analizar en profundidad el trabajo de A.N.Richter y T.M.Khoshgoftaa, vienen dadas por la forma en la que construyen la curva de aprendizaje de los modelos. En su investigación hacen referencia a que construyen la curva tomando el modelo del trabajo de Figueroa y col., pero a la hora de medir el rendimiento de los modelos usan el área bajo la curva de la curva ROC. Pero el modelo presentado en la ecuación (2) únicamente serviría para medir el *Accuracy* de un modelo, por lo que al no presentar ninguna demostración que avale el uso que hacen del modelo, cabe tomar los resultados con cautela.

2.2. Optimización de hiperparámetros

Para resolver el problema de la optimización de hiperparámetros en aprendizaje automático, se han desarrollado numerosas técnicas. A continuación, se hace una introducción a cuatro de los enfoques más utilizados.

2.2.1. Búsqueda en cuadrícula

El método de búsqueda en cuadrícula, o *grid search*, se introdujo por primera vez en 1974 [12], y es uno de los enfoques más básicos para automatizar la optimización de hiperparámetros. Este método consiste en probar mediante fuerza bruta todas las posibilidades dentro de un grupo de valores discretos definidos por el usuario. Se entrena un modelo con cada posible combinación de valores y se evalúan los modelos generados. Por lo que la combinación de hiperparámetros que logra mayor rendimiento en el conjunto de validación, se devuelve como la configuración óptima. El método garantiza encontrar el óptimo global en el conjunto de hiperparámetros predeterminado. Sin embargo, el mayor problema de este enfoque es el alto coste computacional requerido. Es por ello en la literatura se han propuesto diversos formas para optimizar este método [13].

Analizando este método, se ve que el algoritmo toma como entrada un conjunto de hiperparámetros a optimizar $\{h_1, ..., h_k\}$ y para cada uno de ellos un conjunto V de i valores posibles $\{v_{1_i}, ..., v_{n_i}\}$. Por lo que el número de posibles combinaciones es de:

$$\prod_{i=1}^{k} |V_i| = \prod_{i=1}^{k} n_i \tag{3}$$

Por lo tanto, es evidente que el número de configuraciones a probar crece exponencialmente con el número de hiperparámetros y los posibles valores que pueden tomar estos. Si además, se tiene en cuenta que algunos de estos hiperpárametros, al ser continuos, pueden tomar "infinitos" valores, el problema se complica aún más. Es por ello que se dice que este método sufre la maldición de la dimensionalidad. Pero a pesar de la ineficiencia computacional de este enfoque, ha sido el método de optimización de hiperparámetros más popular durante años. En [14] se puede ver que de los artículos aceptados

en NIPS [1] en el año 2014, entre los 86 que hacían uso de alguna técnica de optimización de hiperparámetros, 82 de ellos usaban la búsqueda en cuadrícula. Pero a medida que se han desarrollado nuevos métodos más efectivos, la búsqueda en cuadrícula ha ido quedándose en un segundo plano.

2.2.2. Búsqueda aleatoria

La búsqueda aleatoria [15] se presenta como una evolución del método de búsqueda en cuadrícula. Esta técnica muestrea valores seleccionados aleatoriamente dentro del dominio definido para los hiperparámetros. Es decir, en lugar de definir un conjunto de puntos para cada hiperparámetro, el usuario define un rango donde buscar valores para estos parámetros. En [15] Bergstra y Bengio demuestran que la búsqueda aleatoria es más eficiente en complejidad computacional en comparación con la búsqueda en cuadrícula, y a su vez, que la calidad de los resultados obtenidos son en su mayoría semejantes y, en ciertos casos, superiores.

Una de las razones del éxito de este método, en comparación con el *grid search*, radica en la eficacia que demuestra al trabajar en espacios de alta dimensionalidad. Para un número suficientemente alto de dimensiones, es muy poco probable que todos los hiperparámetros tengan el mismo impacto en la métrica a optimizar. Hay que tener en cuenta que, en la práctica, para hacer posible la búsqueda en cuadrícula se explora un número determinado de valores para cada hiperparámetro, y que la búsqueda aleatoria probablemente probará un nuevo valor para cada hiperparámetro en cada ensayo, puesto que se seleccionan aleatoriamente. Por lo tanto, está claro que la búsqueda en cuadrícula asigna muchos más recursos para la exploración de dimensiones irrelevantes que la búsqueda aleatoria.

En la Figura 2 tomada de [15], se muestra de manera gráfica la observación anterior. En este ejemplo bidimensional, un hiperparámetro tiene una influencia significativamente mayor en el resultado, esto puede verse en las distribuciones marginales representadas en los márgenes de los cuadrados. Mientras que la búsqueda de cuadrícula espaciada uniformemente solo tiene tres valores de ensayo diferentes para el hiperparámetro con mayor importancia, la búsqueda aleatoria da como resultado nueve valores de ensayo

[1] `https://nips.cc/`

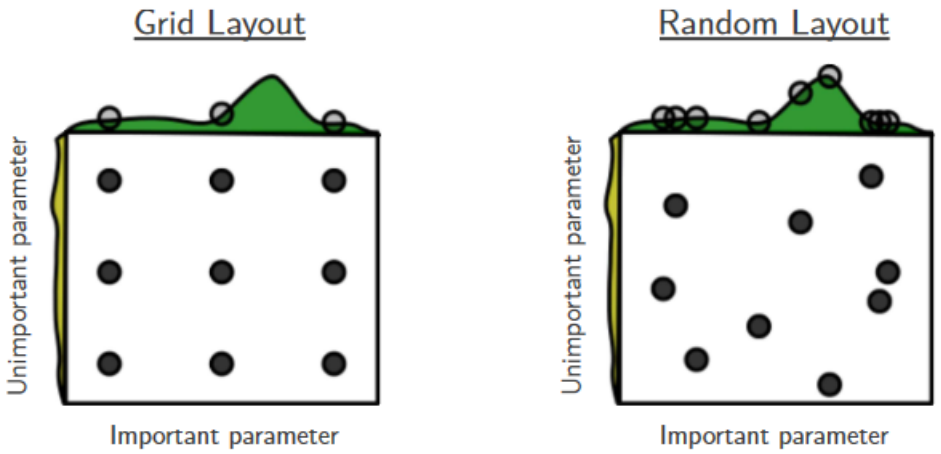


Figura 2: Comparación entre la búsqueda en cuadrícula y aleatoria [15].

diferentes, pudiendo identificar un valor más prometedor para esa dimensión y, por lo tanto, para la configuración de los hiperparámetros.

Aunque podría parecer que un usuario con conocimientos previos podría definir una cuadrícula más apropiada para obtener mejores resultados, Bergstra y Bengio muestran en [15] que incluso para un algoritmo fijo que se evalúa en conjuntos de datos similares, la importancia de diferentes hiperparámetros en el resultado final varía considerablemente, lo que dificulta este trabajo.

2.2.3. Optimización bayesiana

La optimización bayesiana, es un método muy eficaz para resolver problemas de optimización complejos desde el punto de vista computacional [16] [17]. También es conocida como optimización basada en modelos secuenciales o método de configuración adaptativo. Este método se deriva del teorema de Bayes [18], y realiza sus predicciones con una función que es significativamente menos costosa de evaluar.

De manera intuitiva, el método trata el problema como un problema de regresión. Intenta estimar la distribución del rendimiento del algoritmo en función de sus hiperparámetros. Para ello, asume una distribución inicial que modifica iterativamente basándose en los resultados obtenidos con cada com-

binación probada. El objetivo del método es encontrar la configuración que maximice dicha distribución, por lo que debe encontrar un equilibrio entre la exploración y la explotación. Es decir, tiene que hacer uso del conocimiento sobre las áreas donde se estima encontrar los mejores hiperparámetros, pero al mismo tiempo debe explorar las áreas donde todavía no se conocen los resultados. Su funcionamiento se puede describir con la siguiente cadena de ejecuciones:

1. En primer lugar, se define la forma en la que se evaluarán la calidad de los hiperparámetros seleccionados, normalmente se suele utilizar una validación cruzada. Debido a que se desconoce la función objetivo (efectividad del modelo en función de los hiperparámetros), se asume esta como una función aleatoria, a la cual se le especifica una distribución previa. A menudo se utiliza la distribución gaussiana multivariable como distribución previa.

2. Sobre la base de las evidencias que se van logrando al testear diferentes configuraciones de hiperparámetros, se procede a la modificación de dicha distribución. Obteniendo así en cada paso, una distribución más fiel a la realidad.

3. Con la nueva distribución lograda, se decide cual será la siguiente configuración de hiperparámetros a seleccionar según una función de adquisición. Existen diferentes funciones de adquisición definidas, cada cual basa sus elecciones en distintos factores buscando el equilibrio entre exploración y explotación.

4. Se evalúa la configuración seleccionada en el paso 3. Si se cumplen las condiciones de finalización se termina, en caso contrario se vuelve al 2. paso.

Tal y como se ha avanzado en la introducción del presente trabajo, la optimización bayesiana ha demostrado que supera a otros algoritmos de optimización globales en una serie de funciones de referencia de optimización [4], por lo que se ha convertido en uno de los algoritmos más recurrentes en la práctica actual.

En cualquier caso, las tres técnicas presentadas hasta el momento, permiten una gran capacidad de paralelización. Es decir, la capacidad de subdividir

las tareas de búsqueda en distintos servidores o procesadores de forma en que los tiempos de cómputo se reducen considerablemente [2]. Evidentemente, si la búsqueda se realiza en 10 servidores al mismo tiempo, el proceso finalizará mucho más rápido que procesándolo únicamente en un solo ordenador. Y tal y como se afirma en [19], aprovechar los recursos computacionales distribuidos y paralelos se ha vuelto esencial para resolver problemas de alto costo computacional.

2.2.4. Optimización evolutiva

Por último, en este apartado se presenta otro método de optimización de hiperparámetros que en los últimos años está tomando bastante fuerza. La computación evolutiva es una rama de la inteligencia artificial que aborda la resolución automática de problemas de optimización [20]. Para ello, toma la base de inspiración en la evolución biológica con el objetivo de crear algoritmos (algoritmos evolutivos) que permiten encontrar soluciones óptimas o semi-óptimas de forma eficiente. En la optimización de hiperparámetros de un algoritmo de aprendizaje automático dado, el funcionamiento de este método de optimización puede resumirse en los siguientes pasos: [20]

1. Establecido un algoritmo de aprendizaje automático y los hiperparámetros a optimizar de éste, se crea aleatoriamente un conjunto de individuos (o soluciones candidatas). Cada uno de estos individuos será un vector con una dimensión igual al número de hiperparámetros a seleccionar. Por lo que cada posición del vector hará referencia a un hiperparámetro concreto.

2. Se evalúa el rendimiento del algoritmo al ser entrenado con la configuración de cada individuo de la población. Por lo que el grado de adaptación de cada individuo será el rendimiento obtenido con dicha combinación.

3. Utilizando como criterio el valor del grado de adaptación, se selecciona un subconjunto de soluciones candidatas que despeñarán el papel de generar la descendencia.

[2] `https://docs.microsoft.com/es-es/azure/machine-learning/how-to-tune-hyperparameters`

4. La descendencia se obtiene aplicando los operadores de variación adecuados al subconjunto seleccionado en el punto anterior. Y los nuevos individuos así obtenidos compiten con los antiguos para obtener un lugar en la población de la siguiente generación. Esta competición se realiza sobre la base de los valores de adaptación de cada individuo.

5. Los pasos 2-4 se repiten hasta encontrar una solución candidata con suficiente calidad, o hasta que se alcanza una condición de terminación establecida previamente.

En los últimos años se ha extendido el uso de este tipo de técnicas en la optimización de hiperparámetros, teniendo bastante repercusión los trabajos dirigidos a estudiar la optimización de la topología e hiperparámetros de los modelos de *Deep Learning* [21].

2.3. Mejora de la optimización bayesiana

Tal y como se ha expuesto anteriormente, la optimización bayesiana es uno de los métodos que mejores resultados ha mostrado en la tarea de optimizar los hiperparámetros [4]. Es por ello que, en la literatura podemos encontrar diversas investigaciones en las que se ha trabajado en encontrar métodos de hacer más rápida la optimización bayesiana. La razón fundamental de querer acelerar más un método de por si rápido (en comparación con sus competidores), es que cada vez es más común enfrentarnos a situaciones en las que hay que trabajar con mayores cantidades de datos. Por lo que, dependiendo de la escala, un método que en términos relativos puede ser rápido en términos absolutos hace que sea inservible [22], puesto que la cantidad de tiempo necesario para buscar hiperparámetros óptimos crece con bastante rapidez a medida que aumenta el espacio de configuración y el número de datos. Por lo que esto plantea un desafío para el uso del aprendizaje automático en la era del *Big Data*.

En 2017, X. Zeng y G. Luo [23] presentan el método de optimización bayesiana progresiva basada en muestreo, que como su propio nombre indica, combina la optimización bayesiana con el muestreo progresivo. En este caso, el método presentado es capaz de realizar una selección eficiente y automática de algoritmos y valores de hiperparámetros. Y el concepto en el que se basa es en realizar pruebas rápidas en muestras limitadas del conjunto de

datos para eliminar tantas combinaciones poco prometedoras tan rápido como sea posible y dar más recursos para ajustar las prometedoras durante el proceso de optimización bayesiana. El proceso avanza por rondas, en las que se aumenta la base de datos de entrenamiento progresivamente para evitar entrenar las configuraciones poco esperanzadoras con un gran número de datos. Al comparar su método con un método de selección automático de última generación, demuestran que su método obtiene mejores resultados. Y aunque es evidente que el método propuesto ahorra un gran número de cálculos, sigue necesitando invertir un alto tiempo de cómputo con aquellas configuraciones que son entrenadas con bases de datos grandes. Por ello, se aprecia un margen de mejora en este aspecto.

A su vez, ese mismo año A. Klein y col. [24] presentan un método para acelerar la optimización bayesiana al trabajar con grandes bases de datos. En este caso, basan el método en la forma de trabajo de los expertos. Normalmente, al tener que enfrentarse manualmente al trabajo de optimizar los hiperparámetros de un algoritmo en entornos de big data, los expertos realizan una exploración de configuraciones preliminares en subconjuntos pequeños para familiarizarse con los comportamientos de los hiperparámetros y posteriormente extrapolarlo al conjunto completo de datos. Con el objetivo de incorporar ese mecanismo de trabajo dentro de la optimización bayesiana, los autores del trabajo proponen utilizar el tamaño del conjunto de datos como un grado adicional de libertad que enriquezca el proceso de optimización. Es decir, incorporan el tamaño del conjunto de datos con el que realizar el entrenamiento como un pseudo-parámetro de la optimización bayesiana, modelando así el rendimiento y el tiempo de entrenamiento en función del tamaño del conjunto de datos. Por ello, obliga al método a encontrar un equilibrio entre la ganancia de información sobre el óptimo global y el costo computacional. Aunque tal y como demuestran los resultados obtenidos, este método es capaz de reducir el tiempo de cómputo necesario, en ocasiones se sigue necesitando enfrentar al algoritmo en cuestión a ser entrenado con grandes conjuntos de datos ralentizando así el proceso.

Por otro lado, con el objetivo de acelerar la optimización bayesiana en la selección de hiperparámetros de redes neuronales, en 2018 D.Choi y col. [25] presentaron un trabajo en el que estudiaban como predecir y utilizar las curvas de aprendizaje de estos algoritmos de aprendizaje automático para finalizar antes de tiempo la fase de entrenamiento de las configuraciones poco

prometedoras. Pero en este caso, a la hora de construir las curvas de aprendizaje, en vez de medir el rendimiento en función del tamaño muestral (tal y como se explica en [5]), lo miden en función del las épocas de entrenamiento del modelo. Esta puede ser una de las razones por las que los resultados del estudio no son del todo satisfactorios, al no encontrar patrones consistentes en las curvas de aprendizaje.

2.4. Conclusiones

En este apartado, se pretende presentar un resumen de los aspectos fundamentales desprendidos del análisis del estado del arte realizado, y a su vez, presentar como conclusión la hipótesis de la presente investigación.

Al comienzo de este capítulo, se ha visto como diversas investigaciones han demostrado la capacidad del método de estimación de curvas de aprendizaje para predecir el rendimiento de un modelo de aprendizaje automático. En segundo lugar, se destaca la demostración empírica en investigaciones de la superioridad de la optimización bayesiana sobre otros métodos de optimización de hiperparámetros. Por último, se ha visto como los diversos métodos propuestos en la literatura para la mejora de la eficiencia de la optimización bayesiana tienen margen de mejora. Precisamente, las diversas alternativas estudiadas, no dejan de necesitar entrenar, en algunos casos, el modelo de aprendizaje automático con gran parte de la base de datos, cosa que en entornos de *Big Data* empeora el costo computacional.

Teniendo en cuenta estos tres aspectos, se propone como hipótesis que, haciendo uso de la estimación por curvas de aprendizaje se pueda hacer más eficiente la optimización bayesiana. Y esto se debería, a que, sustituyendo la estimación mediante validación cruzada por la metodología propuesta, se podría obtener una estimación de rendimiento del algoritmo, entrenándolo únicamente con una pequeña cantidad de datos. Por lo que, en términos generales, demostrar la veracidad de esta hipótesis podría suponer un importante avance en el campo de la optimización de hiperparámetros en entornos de *Big Data*.

3. Métodos empleados

En este capítulo se presentan y se analizan los métodos utilizados en la fase experimental de la investigación. Comenzando por las curvas de aprendizaje, seguido de la optimización bayesiana y finalizando con el algoritmo diseñado para la combinación de ambos métodos.

3.1. Curvas de aprendizaje

A continuación se expone el método para la predicción de curva de aprendizaje propuesto por Figueroa y col. tomando como referencia la notación del artículo [5].

Tal y como se ha introducido al analizar el estado del arte del estudio de las curvas de aprendizaje, la curva de aprendizaje de un modelo predictivo es una gráfica en la que se enfrentan el tamaño muestral con el que se entrena el modelo, con el rendimiento del modelo obtenido con dicha cantidad de datos de entrenamiento.

En investigaciones realizadas, se ha comprobado que generalmente las curvas de aprendizaje siguen la ley de potencia inversa [11]. Según esta ley, el error y el *accuracy*, respectivamente, quedarían así representados en función del tamaño muestral con el que un modelo haya sido entrenado:

$$e(n) = a + b \cdot n^c, \tag{4}$$

$$A(n) = (1 - a) - b \cdot n^c, \tag{5}$$

donde:
$0 < E[\ a\ ] \ll 1$ y $E[\ c\ ] \in [-1, 0]$ siendo,
$a \equiv$ *error mínimo alcanzable;* $b \equiv$ *tasa de aprendizaje;* $c \equiv$ *tasa de caída.* [5]

En este caso se representa el *accuracy* porque en esta investigación se trabaja con problemas de clasificación. Pero con la función del error se podría trabajar sin problemas en problemas de regresión.

Veamos ahora qué procedimiento seguir para obtener la curva de aprendizaje en cada caso. En primer lugar, se asume que se quiere ajustar una

curva de aprendizaje que representa el *accuracy* (Y_{acc}) de un determinado clasificador en función el tamaño muestral con el que ha sido entrenado (X). Con este par de conjuntos se realizará una regresión basada en mínimos cuadrados no lineales [6], con el objetivo de obtener los parámetros a, b y c de la ecuación (5) que mejor se adapten a la nube de puntos dada.

Para poder realizar la regresión de la curva, son necesarios un conjunto de puntos iniciales de esta. Para obtener dicha muestra de valores (x_j, y_j), se toma una muestra incremental del tipo $\overrightarrow{x_j} = \{k, 2 \cdot k, \ldots, m \cdot k\}$ donde $k > 0$ determina el tamaño de salto que habrá entre los distintos valores del conjunto $\overrightarrow{x_j}$, y m determina el número de puntos utilizados para estimar la curva de aprendizaje. Entonces, para cada uno de los $x_j \in \overrightarrow{x_j}$ se entrena el clasificador con un conjunto de datos de tamaño x_j. Tras entrenar el clasificador con su correspondiente tamaño muestral, se evalúa el *accuracy* de dicho modelo para obtener así el valor de y_j. Tras repetir el proceso con todos los valores de $\overrightarrow{x_j}$, se habrá obtenido el conjunto $\overrightarrow{y_j}$. La evaluación se puede realizar usando un conjunto de test independiente o mediante un *cross-validation*. Los vectores $\overrightarrow{x_j}$ e $\overrightarrow{y_j}$ muestran los puntos iniciales con los que estimar la curva de aprendizaje.

En segundo lugar, se define Ω como una colección de puntos de una curva de aprendizaje empírica correspondientes a $(\overrightarrow{x_j}, \overrightarrow{y_j})$. En caso de querer medir la calidad de la curva de aprendizaje estimada, Ω puede ser dividido en dos subconjuntos Ω_t y Ω_v, uno para realizar la regresión y el otro para evaluar la calidad de dicha regresión. Para ello se fija un $x_s \in \overrightarrow{x_j}$ tal que $\Omega_t = \{(x_j, y_j | x_j \leq x_s)\}$ y $\Omega_v = \{(x_j, y_j | x_j > x_s)\}$. En caso de no querer realizar dicha división, se asumirá que $\Omega_t = \Omega$.

Una de las principales novedades propuestas por Figueroa y col. en su trabajo, consiste en asignar pesos a los diferentes puntos del conjunto $(\overrightarrow{x_j}, \overrightarrow{y_j})$, con el objetivo de dar más importancia a unos puntos que a otros a la hora de realizar la regresión. Para asignar pesos a los puntos de Ω_t se asume que el desempeño del clasificador en un tamaño de muestra de entrenamiento más grande es más indicativo del desempeño futuro del clasificador. Por lo que a cada punto de la curva $(x_j, y_j) \in \Omega_t$ se le asigna un peso normalizado $\frac{j}{m}$ donde $m = |\Omega_t|$.

Posteriormente, usando el conjunto Ω_t se aplica una optimización no lineal de mínimos cuadrados ponderados para ajustar el modelo de la ecuación (5) encontrando los parámetros $\{a, b, c\}$. Por último, en caso de haber dividido el conjunto Ω en Ω_t y Ω_v para evaluar la calidad de predicción de la curva, se calcula el error absoluto medio (EAM) o el error cuadrático medio (ECM) entre los puntos de Ω_v y las predicciones de la curva para esos mismos puntos.

$$EAM = \frac{1}{|\Omega_v|} \sum_{(x_j,y_j)\in\Omega_v}^{m} |y_j - \hat{y_j}|, \ \forall (x_j, y_j) \in \Omega_v \tag{6}$$

$$ECM = \sqrt{\frac{\sum_{(x_j,y_j)\in\Omega_v}^{m} (y_j - \hat{y_j})^2}{|\Omega_v|}}, \ \forall (x_j, y_j) \in \Omega_v \tag{7}$$

En el **apéndice A**, se recoge el código desarrollado por Figueroa y col. en su trabajo [5] en el que se implementa el proceso aquí explicado en el lenguaje `R`.

3.1.1. Mínimos cuadrados no lineales

En este apartado se presenta el método de los mínimos cuadrados no lineales, el cual se utiliza en el ajuste del modelo estándar de la curva de aprendizaje a los datos.

La técnica de los mínimos cuadrados no lineales surge de la necesidad de aproximar a formulaciones estándar el comportamiento de sistemas económicos, sociales, físicos... que por su complejidad necesitan de representaciones no lineales [26]. Por lo tanto, de forma más concreta, podría decirse que la técnica de mínimos cuadrados no lineales es la forma de ajustar un modelo no lineal de n parámetros desconocidos a un conjunto de m observaciones, donde $m \geqslant n$.

Antes de nada, hay que tener en cuenta que en esta situación, un modelo es considerado no lineal cuando no lo es respecto a los parámetros, es decir, no importa que las variables del modelo sean o no lineales. Por lo tanto, se

dirá que un modelo es no lineal cuando no haya una manera expresar una relación lineal entre los parámetros [27]. Por ejemplo:

1. $y = \beta_1 \cdot x_1^{\beta_2} \cdot x_2^{\beta_3} \cdot \mu$ puede expresarse de forma lineal al hacer $ln(y) = ln(\beta_1) + ln(\beta_2)x_1 + ln(\beta_3)x_2 + ln(\mu)$. Por lo que es lineal en los parámetros.

2. $y = \beta_1 \cdot x_1^{\beta_2} \cdot x_2^{\beta_3} + \mu$ en cambio, no puede expresarse de forma lineal, por lo que es no lineal en los parámetros.

Sea pues, un conjunto de m observaciones $\{(x_1, y_1), ..., (x_m, y_m)\}$ y una función $y = f(x, \beta)$ no lineal respecto a sus n parámetros $\beta = (\beta_1, ..., \beta_n)$: $m \geqslant n$. Por lo tanto, la técnica de mínimos cuadrados no lineales busca encontrar los valores del vector β que mejor ajusten la función $y = f(x, \beta)$ a la nube de puntos dada en el conjunto de las observaciones. Para ello, plantea minimizar la siguiente expresión [27] [28] [29]:

$$S = \frac{1}{2} \sum_{i=1}^{m} r_i^2 \ : \ r_i = y_i - f(x_i, \beta) \tag{8}$$

Por lo tanto, el problema de minimización se puede plantear en los siguientes términos:

$$min_{\beta \in B} \ S(\beta) = min_{\beta \in B} \ \frac{1}{2} \sum_{i=1}^{m} r_i^2(\beta) = min_{\beta \in B} \ \frac{1}{2} \|r(\beta)\|_2^2 \tag{9}$$

Para resolver este problema, se plantea un proceso iterativo de aproximación a la solución en la que $r(\beta)$ se modeliza en cada paso en función del desarrollo de Tylor hasta su primera derivada [26]. Es decir,

$$r(\beta) = r(\beta^k) + J(\beta^k) \cdot (\beta - \beta^k) \tag{10}$$

Donde J hace referencia a la matriz Jacobiana y k al número de iteración.

Por lo que, el proceso iterativo planteado puede resumirse en los siguientes pasos [26]:

1. Se define un β^0 inicial y se toman $k = 1$ y $\beta^k = \beta^0$.

2. Se determina el $min_{\beta \in B} \ \frac{1}{2} \|r(\beta^k) + J(\beta^k) \cdot (\beta - \beta^k)\|_2^2$.

3. En caso de que $\beta - \beta^k < tolerancia$ (para un valor de *toleracia* preestablecido), se da por resulto el problema. En caso contrario, $k = k + 1$, $\beta^k = \beta$ y se vuelve al segundo paso.

El subproblema planteado en el segundo paso es un problema lineal del tipo:

$$min_{X \in \mathcal{R}^n} \ \|AX - b\|_2, \tag{11}$$

por lo que puede resolverse fácilmente con métodos conocidos para mínimos cuadrados lineales: Ecuaciones normales, factorización QR, descomposición en valores singulares...

3.2. Optimización bayesiana

Este apartado ha sido desarrollado tomando como referencia los trabajos: [30] [31]. La optimización bayesiana, es una técnica de optimización basada en aprendizaje automático y con un gran componente matemático, enfocada en la optimización de una función objetivo $f(x)$ dada. La expresión matemática correspondiente es la siguiente:

$$x_* = opt_{x \in \mathbf{X}} \ f(x) \ : \ \mathbf{X} \equiv espacio\ de\ búsqueda \tag{12}$$

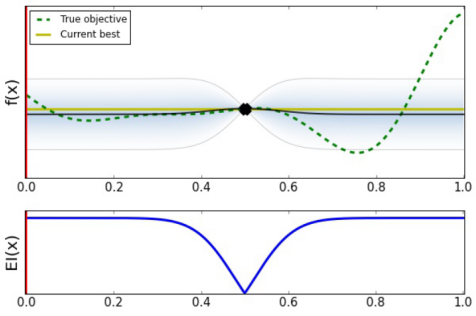
Esta técnica fue diseñada para ser aplicada en situaciones con las siguientes características:

- $f(x)$ es una función desconocida.
- $f(x)$ es muy costosa en términos computacionales.
- La forma de lograr observaciones de $f(x)$ no pasa por trabajar con las derivadas de esta función.
- Espacios de búsqueda con menos de 20 dimensiones.

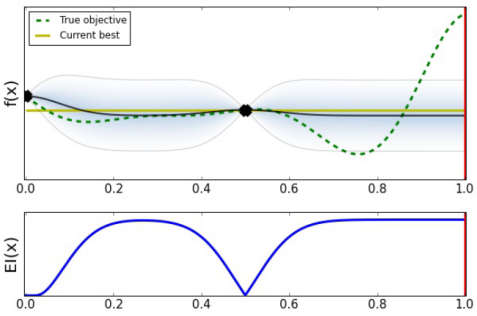
A grandes rasgos, la optimización bayesiana se centra en encontrar los valores de $x \in \mathbf{X}$ que además de dar la mayor cantidad de información sobre la distribución de $f(x)$, optimicen la función realizando el menor número de observaciones. La ventaja principal de este método, es que es capaz de mantener un equilibrio entre la explotación y la exploración del espacio de

búsqueda X. Mientras que la exploración se centra en explorar áreas de X desconocidas, la explotación se centra en la búsqueda de nuevas soluciones en las inmediaciones de las soluciones más prometedoras obtenidas hasta el momento.

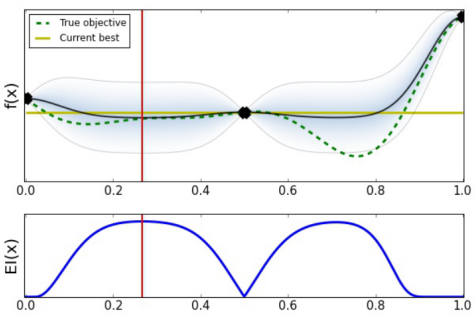
La optimización bayesiana está construida sobre dos componentes principales: un modelo estadístico bayesiano para la modelización de la función objetivo, y una *función de adquisición* (ver subapartado 3.2.2) capaz de decidir donde realizar la siguiente búsqueda en X. El modelo estadístico, el cual es un *proceso gaussiano* (ver subapartado 3.2.1) proporciona una distribución de probabilidad que describe los valores potenciales de $f(x)$ para cada valor de x. Por lo que, cada vez que se observa $f(x)$ en un punto, esta distribución de probabilidad es actualizada. Por otro lado, la función de adquisición se encarga de estimar el valor que debería ser generado al evaluar la función objetivo en un nuevo punto x. Para ello, hace uso de la distribución de probabilidad obtenida por el proceso gaussiano. Por lo tanto, está claro que la optimización bayesiana se trata de un proceso iterativo de optimización. En las siguientes ilustraciones tomadas de [32] se puede ver en forma visual como realiza la optimización bayesiana el trabajo de minimización. En cada iteración, en la figura superior se muestran: la función objetivo $f(x)$ desconocida (curva discontinua verde), evaluaciones puntuales de $f(x)$ realizadas (cruces negras), la función estimada $\hat{f}(x)$ (curva negra), mejor valor obtenido hasta el momento (linea verde claro) y la distribución de probabilidad de $f(x)$ obtenida en el proceso gaussiano (entorno gris). Y en la figura inferior: la función de adquisición asociada a la correspondiente distribución de probabilidad gaussiana (curva azul) y el máximo de dicha curva, y por lo tanto, próximo punto a estimar(linea vertical roja).



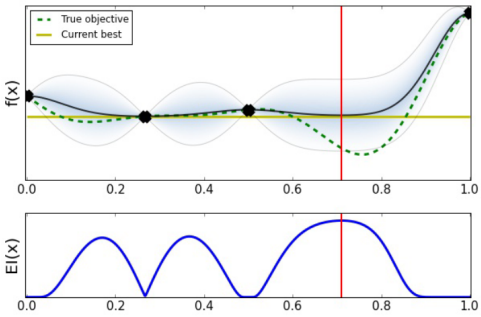
1. Iteración



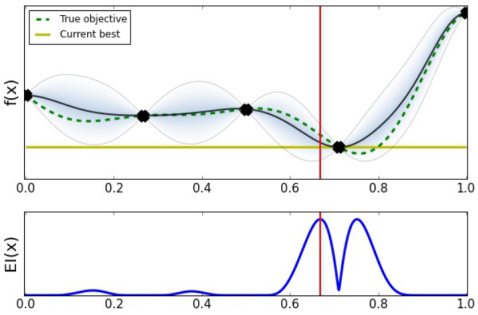
2. Iteración



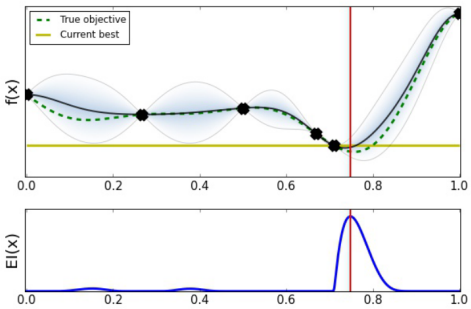
3. Iteración



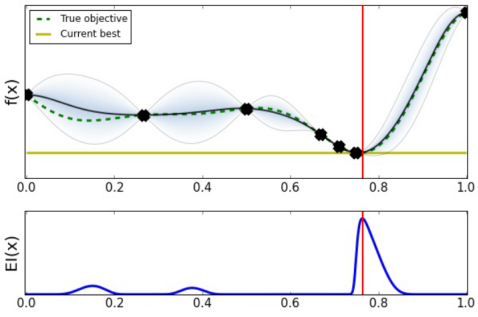
4. Iteración



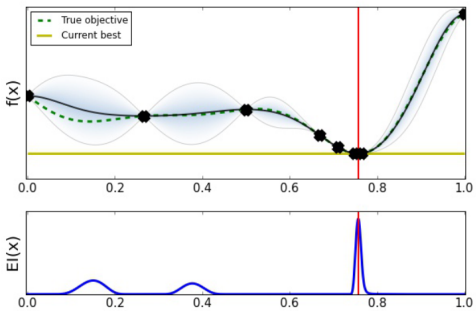
5. Iteración



6. Iteración



7. Iteración



8. Iteración

3.2.1. Procesos gaussianos

El proceso gaussiano es un método de estadística bayesiana para la modelización de funciones. Suponiendo que el objetivo sea modelizar una función $f(x)$ dada y que se conocen los valores de $\{x_1, ..., x_k\}$ para la misma; se asume que el vector $[f(x_1), ..., f(x_k)]$ sigue una distribución normal multivariable con vector media y matriz de covarianzas asociados. Se asume a su vez, que tanto el vector de medias como la matriz de covarianzas son el resultado de aplicar respectivamente la función media μ_0 a cada punto x_i y la función covarianza o kernel a cada par de puntos (x_i, x_j). Por lo tanto tenemos que:

$$f(x_{1:k}) \sim \mathcal{N}(\mu_0(x_{1:k}), \Sigma_0(x_{1:k}, x_{1:k})), \tag{13}$$

donde $x_{1:k}$ hace referencia a la secuencia $\{x_1, ..., x_k\}$.

Tomando como punto de partida lo expuesto hasta ahora, suponemos que deseamos inferir la nueva distribución de probabilidad tras evaluar la función en un nuevo punto x_{k+1}. Para ello, debemos calcular esta nueva distribución condicionada haciendo uso de la regla de bayes:

$$f(x)|f(x_{1:k}) \sim \mathcal{N}(\mu_k(x), \sigma_k^2(x)), \tag{14}$$

donde:

$$\mu_k(x) = \Sigma_0(x, x_{1:k})\Sigma_0(x_{1:k}, x_{1:k})^{-1}(f(x_{1:k}) - \mu_0(x_{1:k})) + \mu_0(x) \tag{15}$$

$$\sigma_k^2(x) = \Sigma_0(x, x) - \Sigma_0(x, x_{1:k})\Sigma_0(x_{1:k}, x_{1:k})^{-1}\Sigma_0(x_{1:k}, x) \tag{16}$$

Por lo tanto, la distribución condicional de $f(x)$ dado $f(x_{1:k})$ sigue una distribución normal multivariante. Donde el vector media y el *kernel* de covarianza dependen de la posición del punto sin evaluar, la posición de los puntos evaluados y los valores de dichas evaluaciones. Esta distribución condicionada es llamada distribución de probabilidad posterior.

3.2.1.1. Función media y kernel

Aunque existen más de un tipo de función media que puede ser usada, una de las opciones más utilizadas es tomar la función constante. Es decir,

$$\mu_0(x) = \mu \in \mathrm{R}^p. \tag{17}$$

Cuando se tienen indicios de que la función objetivo $f(x)$ sigue algún tipo de tendencia o algún tipo de aplicación paramétrica concreta, se toma como función de media la siguiente:

$$\mu_0(x) = \mu + \sum_{i=1}^{p} \beta_i \psi_i(x), \tag{18}$$

donde cada ψ_i es una función paramétrica y β_i sus parámetros asociados.

Respecto a la función del *kernel*, hay que tener en cuenta que ésta debe seguir la siguiente lógica: los puntos que estén más cerca deben estar fuertemente correlacionados, este es, en caso de que $\|x - x'\| < \|x - x''\|$ para una norma $\|.\|$ dada, entonces $\Sigma_0(x, x') > \Sigma_0(x, x'')$. A su vez, las matrices *kernel* deben ser positivas.

Una de las funciones kernel más usadas es la *Gaussian kernel:*

$$\Sigma_0(x, x') = \alpha_0 \; exp(-\|x - x'\|^2), \tag{19}$$

donde

$$\|x - x'\|^2 = \sum_{i=1}^{d} \alpha_i (x_i - x_i')^2, \tag{20}$$

siendo $\alpha_{0:d}$ los parámetros del *kernel*.

Tal y como se ha visto, tanto la función media como el *kernel* pueden constar de parámetros. Por ejemplo, en caso de tomar la función media paramétrica y el *kernel Gaussiano*, se tendría el siguiente conjunto de parámetros a definir:

$$\eta = (\beta_{1:p}, \alpha_{0:d})$$

Para la selección de dichos parámetros la aproximación más utilizada es la de tomar el estimador de máxima verosimilitud. Dadas las observaciones $x_{1:n}$ la verosimilitud de estas observaciones se calcula sobre la distribución actual. Por lo que el problema se reduce a encontrar:

$$\eta = argmax_\eta \; P(f(x_{1:n})|\eta). \tag{21}$$

3.2.2. Función de adquisición

Una vez determinado como funciona el primer componente de la optimización bayesiana, falta por analizar el segundo elemento. La función de adquisición, al definir cómo muestrear el espacio de búsqueda es la encargada de mantener el equilibrio entre la explotación y la exploración; por lo que tiene un papel determinante.

Existen múltiples funciones de adquisición, pero las más conocidas y las más utilizadas son dos: la *probability of improvement* (PI) y la *expected improvement* (EI).

La PI mide la probabilidad de que la próxima observación sea mejor que el mejor valor que se haya obtenido hasta el momento. Antes se ha visto como la probabilidad condicionada de $f(x)$ seguía la llamada distribución gaussiana de probabilidad. Por lo tanto, asumiendo un problema de minimización,

$$PI(x) = P(f(x) \leq f(x^+) - \epsilon), \tag{22}$$

siendo $f(x^+)$ el mejor valor obtenido hasta el momento y ϵ un parámetro infinitesimal definido de antemano. Puesto que gracias al proceso gaussiano se tiene una estimación de la distribución de $f(x)$, lo anterior se puede expresar como

$$PI(x) = \Phi(\frac{f(x^+) + \epsilon - \mu(x)}{\sigma(x)}), \tag{23}$$

donde Φ es la función de probabilidad acumulada de la distribución gaussiana, siendo μ y σ, respectivamente, sus funciones media y covarianza asociadas. El problema de la estrategia que sigue *PI*, es que se centra sobre todo en la explotación. Por eso, con el objetivo de tener en cuenta también las zonas no exploradas, se propone la segunda función de adquisición *EI*, la cual calcula la mejora esperada respecto al mejor valor obtenido hasta el momento. Su expresión matemática viene dada por:

$$EI(x) = \begin{cases} (f(x^+) + \epsilon - \mu(x))\Phi(Z) + \sigma(x)\Phi(Z) \ si \ \sigma(x) > 0 \\ 0 \ si \ \sigma(x) = 0 \end{cases} \tag{24}$$

donde $Z = \dfrac{f(x^+) + \epsilon - \mu(x)}{\sigma(x)}$.

Por último, la maximización de la función de adaptación se realiza haciendo uso de métodos numéricos convencionales.

3.3. Algoritmo diseñado

En el anterior apartado (3.2) se han expuesto los conceptos teóricos de la optimización bayesiana. En el área de la optimización de hiperparámetros de algoritmos de aprendizaje automático, éste ha sido un método que ha obtenido resultados destacables en términos de coste computacional. Pero cuando se trata de trabajar con grandes bases de datos, los resultados no son tan favorables. El mayor problema reside en que, el tiempo de cómputo necesario para entrenar un algoritmo y evaluarlo, crece a medida que aumenta el número de datos con los que se trabaja, en ocasiones incluso con relaciones exponenciales. Esto se debe a que normalmente para evaluar la función objetivo ($f(x) \equiv$ *rendimiento del modelo*), suele hacerse uso de la validación cruzada; pero incluso cuando las bases de datos son lo bastante grandes, la técnica de conjuntos *train-test* para la evaluación tiene un coste computacional demasiado elevado.

La propuesta que se realiza en este trabajo es la de utilizar la predicción de curvas de aprendizaje para realizar la evaluación del rendimiento de los modelos; reduciendo así el tiempo de cómputo necesario para evaluar la función objetivo $f(x)$, y por ende, para la optimización bayesiana de hiperparámetros. La lógica reside en que si entrenando los modelos con un pequeño porcentaje de los datos posibles, se es capaz de predecir el rendimiento del modelo al entrenarlo con una determinada cantidad de datos, el coste de evaluar el rendimiento de cada configuración de hiperparámetros se debería reducir notablemente.

En **Algorithm 0** se formaliza el algoritmo con el que se realiza la predicción de la curva de aprendizaje y se devuelve una predicción del rendimiento del modelo estudiado. A continuación, se procede a analizar dicho algoritmo.

Tal y como se puede observar, el algoritmo comienza contando el número de individuos (**N**) de la base de datos (**BBDD**). Posteriormente, establece el número de individuos destinados a la evaluación (**n**) siguiendo la proporción (**P**) deseada y tomando el múltiplo de 10 más cercano inferiormente a dicha proporción. Luego se define la lista de valores de X (**Nube_X**), los cuales

hacen referencia al número de individuos con lo que es entrenado el modelo. Dicha lista de valores, tendrá un valor de inicio (**k1**), un valor final (**k2**) y entre ambos se seleccionarán los valores cada $k3$. Por ejemplo, si se toman k1=5, k2=11 y k3=2 la lista de valores será {5,7,9,11}.

Seguidamente, se establece una lista vacía (**Nube_Y**) donde se deberán almacenar los rendimientos correspondientes a entrenar el modelo con la cantidad de individuos de la lista **Nube_X**. Así pues, con ambos conjuntos se obtendrá la nube de puntos que relaciona el cardinal de datos de entrenamiento con el rendimiento del modelo.

En un siguiente paso, se entra en un bucle "*for*" para cada valor de la lista **Nube_X**. En el que, en primer lugar, se define una lista vacía (**Accuracy**) donde almacenar los rendimientos del resultado de validar el modelo. Ahora, se entra en un nuevo bucle de un número determinado de iteraciones (**S**); donde en cada iteración se entrenará y validará el modelo una vez. Dentro de este último bucle, antes de nada, se seleccionan de forma aleatoria los conjuntos de validación y entrenamiento correspondientes. Hay que tener en cuenta que en cada iteración habrá que seleccionar un número x_t de individuos para entrenar el modelo y n individuos para la validación del mismo. Siendo $x_t + n << N$, donde x_t es el número t-ésimo de la lista de valores **Nube_X**. Después, se entrena, con el conjunto de entrenamiento, un clasificador con el algoritmo de aprendizaje automático definido (**Modelo**) y sus correspondientes hiperparámetros (**H**). Tomando el conjunto de validación, se evalúa el rendimiento del modelo entrenado (**acc**); y se adjunta a la lista de rendimientos (**Accuracy**).

Una vez salido del bucle de las **S** iteraciones, se adjunta en la lista **Nube_Y** la media de la lista de rendimientos (**Accuracy**) obtenida. Este valor medio, será asumido como el rendimiento esperado de entrenar el **Modelo** con los hiperparámetros **H** y con un número de datos x_t.

Por último, una vez salidos del bucle principal, y se ha obtenido la nube de puntos deseada {Nube_X,Nube_Y}; se ajusta la curva de aprendizaje a dicha nube de puntos tal y como se ha estudiado en el apartado 3.1. Finalmente, el algoritmo devuelve el *accuracy* esperado de entrenar el **Modelo**, con los hiperparámetros **H** y con un número determinado (**l**) de datos.

Algorithm 0: Algoritmo LCP(Learning Curve Prediction)

Entrada: Base de datos (**BBDD**); Variable independiente (V); modelo de clasificación binaria definido y no entrenado (**Modelo**); Conjunto de hiperparámetros del Modelo (H); proporción de datos dedicados a la evaluación (P); primer valor del conjunto $\overrightarrow{x_j}$ ($k1$); último valor del conjunto $\overrightarrow{x_j}$ ($k2$); tamaño de los saltos entre los valores $k1$ y $k2$ ($k3$); número de validaciones hechas para calcular el rendimiento con un determinado número de datos de entrenamiento (S); límite en el que queremos conocer la estimación de rendimiento (l).

Salida : Accuracy estimado para el **Modelo** al ser entrenado con una base de datos de tamaño l.

Inicio:

N ← **Numero_Individuos**(**BBDD**);[1]
n ← $(N \cdot P) - mod(N \cdot P, 10)$;
Nube_X ← $\{k1 : k3 : k2\}$;
Nube_Y ← {};
for *x in Nube_X* **do**
    Accuracy← {};
    for *i=1 to S* **do**
        Validacion,Entrenamiento← **Aleatorizar**(**BBDD**, x,n);[2]
        Clasificador ← **Entrenar**(**Modelo**,V,$Entrenamiento, H$);[3]
        acc ← **Evaluar_Accuracy**($Clasificador, Validacion$);[4]
        Accuracy ← Accuracy ∪ acc;
    end
    Nube_Y ← Nube_Y ∪ **Media**(Accuracy);
end
Pesos ← $\{\frac{i}{|Nube\_X|}\}$ **for** i=1 to $|Nube\_X|$;
Curva ← **Ajustar_Curva**($Nube\_X, Nube\_Y, Pesos$);
Devolver: (Curva(l))

Notas:

- [1] La función **Numero_Individuos** calcula el número de individuos de una base de datos dada.
- [2] Con la función **Aleatorizar** se divide la base de datos en un conjunto de entrenamiento y otro de evaluación siendo x para entrenar y n para validar.
- [3] La función **Entrenar**, tal y como su nombre indica, entrena el **Modelo** con los hiperparámteros H dados como input con la base de datos Entrenamiento.
- [4] La función **Evaluar_Accuracy** calcula la precisión del modelo entrenado, evaluándolo con la base de datos Validación.

Por lo tanto, la propuesta realizada en este trabajo, reside en que la optimización bayesiana calcule el rendimiento de cada configuración de hiperparámetros haciendo uso del algoritmo aquí expuesto. Pero antes de pasar a la fase experimental, hay que estudiar el comportamiento del algoritmo para aprender a usarlo de la manera más correcta. En el siguiente subapartado se aborda esta cuestión.

3.3.1. Estudio del algoritmo

En este subapartado, el objetivo ha sido estudiar y sacar conclusiones sobre el uso de distintas configuraciones de parámetros de entrada en el **Algoritmo LCP**. Este trabajo se ha realizado con el objetivo de hacer un mejor uso de dicho algoritmo en la práctica. En concreto, en el algoritmo definido anteriormente, hay varios parámetros que deben ser definidos por el usuario:

- **P**≡ Proporción de datos dedicados a la evaluación.
- **{k1,k2,k3}**≡ Serie de puntos incluidos en el conjunto Nube_X.
- **S**≡ Número de veces que se evalúa el rendimiento de un modelo entrenado con un determinado número de datos.

Por otro lado, el usuario puede decidir también si hacer uso de la base de datos completa, o si tomar una muestra aleatoria de esta para usarla en el algoritmo. En este caso, se ha decidido mantener el valor de **P** fijo en $\frac{1}{3}$ y variar el número de datos incluidos en la base de datos (**N**). La razón de esta elección se debe a que en la práctica, en numerosas ocasiones, las bases de datos no tienen un formato compatible con el uso de este tipo de algoritmos. Por lo que se debe realizar una extracción y conversión a un formato compatible. Al realizar esta acción en entornos de grandes bases de datos, se debe seleccionar un número de datos a convertir, pudiendo seleccionar toda la base de datos o parte de ella.

Tanto en este apartado como en la fase experimental, se ha querido trabajar con dos modelos de aprendizaje automático. Se han seleccionado un modelo del área del aprendizaje automático clásico y otro del área del *deep learning*. Del área del aprendizaje automático ha sido seleccionado el algoritmo de clasificación SVM [33]; no solo por los buenos resultados que ha mostrado, sino también por el coste computacional tan elevado que tiene al trabajar con grandes bases de datos. Del área del *deep learning*, ha sido seleccionado la red neuronal del tipo perceptrón multicapa (RN) [34] al ser un modelo fundamental en esta área y a su vez de los más utilizados.

En esta fase de estudio se ha trabajado con dos bases de datos. En este

caso, se ha recurrido a un repositorio público de datos[3], y las características que han guiado la selección de bases de datos han sido las siguientes: que tuvieran algunas decenas de miles de individuos, que no tuvieran un exceso de valores perdidos, que tuviesen recogidas algunas decenas de variables y que existiera una variable categórica binaria que pudiese ser tomada como variable a predecir. A continuación, se muestra un resumen de los dos conjuntos seleccionados, los cuales cumplen las características aquí mencionadas.

Adult Dataset[4] :

Esta base de datos recoge información sobre distintos aspectos de individuos adultos estadounidenses. Como por ejemplo: la edad, el sexo, nivel educativo, país de origen ... Y entre otras, también encontramos el rango salarial de cada persona, el cual puede estar por encima de los 50000 dolares anuales o por debajo. En este caso, se ha tomado esta última variable como la variable dependiente a predecir por los modelos. En total la base de datos cuenta con 15 variables, 9 de ellas categóricas y 6 numéricas; e información sobre 32561 individuos. Todas las variables categóricas se han convertido en numéricas para poder ser utilizadas por los algoritmos de clasificación SVM y RN.

CelebFaces Dataset[5] :

En este caso, estamos ante un conjunto de datos ideado para entrenar y probar modelos para la detección de rostros. En concreto, en la base de datos usada, cada individuo hace referencia a una fotografía; y las distintas variables nos dan información sobre las personas que aparecen en ellas: color de pelo, color de ojos, si llevan sombrero, si están sonriendo ... En total contamos con 40 variables categóricas, y en el caso de los modelos del tipo SVM se ha usado como variable dependiente la variable binaria que informa sobre si la persona fotografiada usa o no pintalabios; y en el caso de los modelos del tipo RN la variable binaria que informa sobre si el individuo lleva o no corbata. La base de datos reúne información sobre 202599 retratos, y como en el caso anterior, las variables categóricas han sido reconvertidas a

[3] `https://www.kaggle.com/`

[4] `https://www.kaggle.com/kashnitsky/mlcourse?select=adult.data.csv`

[5] `https://www.kaggle.com/jessicali9530/celeba-dataset`

numéricas para poder ser usadas por el SVM y RN.

3.3.1.1. Modelos con SVM

En el **Apéndice B** se encuentra el código en lenguaje `R` de la función programada para realizar las predicciones de la curva de aprendizaje con modelos de SVM. Puesto que el objetivo en este punto se centra en estudiar como hacer uso de los parámetros del **Algoritmo LCP**, se decide fijar unos valores para los hiperparámetros del SVM. A continuación se muestra la configuración seleccionada:

Type	'C-classification'
Kernel	'polynomial'
Degree	3
Gamma	1/15
Coef0	0
Cost	1
Tolerance	0.001
epsilon	0.1
shrinking	1

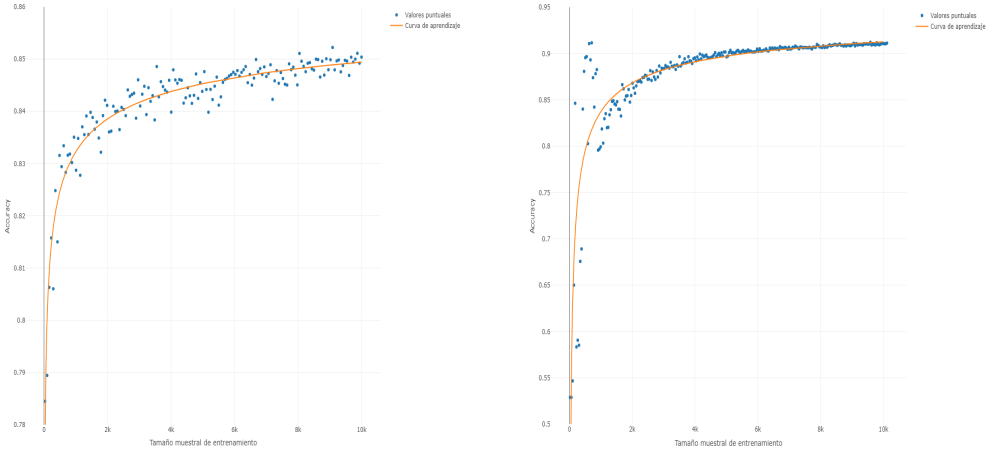
Tabla 1: Hiperparámetros del algoritmo SVM fijados.

◦ Curvas de aprendizaje esperadas:

En primer lugar, se han calculado las que serán asumidas como las curvas de aprendizaje esperadas al aplicar el algoritmo SVM con los hiperparámetros recogidos en la Tabla 1 a las bases de datos *Adult* y *CelebFaces*. Para ello, se ha seguido el mismo procedimiento marcado por el **Algoritmo LCP** para obtener la nube de puntos {Nube_X , Nube_Y}. Se han tomado una gran cantidad de valores de X en el rango [10,10000], con el objetivo de poder estimar la curva de aprendizaje de la manera más precisa posible.

Una vez obtenida la nube de puntos, se ha realizado la regresión de mínimos cuadrados sobre dicho conjunto de puntos. En las gráficas que se muestran a continuación se pueden observar en color azul las nubes de puntos obtenidas, y en color naranja las curvas de aprendizaje que se ha estimado sobre dichas nubes. Por lo tanto, dada la gran cantidad de puntos con los

que han sido calculadas las curvas, éstas serán asumidas como las curvas de aprendizaje esperadas.



a: Adult b: CelebFaces

Figura 7: Curvas de aprendizaje esperadas con SVM.

A continuación, se muestran los resultados numéricos del proceso de mínimos cuadrados no lineales aplicado. Nótese que el proceso de mínimos cuadrados no lineales permite obtener la significatividad estadística de los parámetros de la curva estimados, lo que permite cuantificar la calidad de la estimación.

Adult:

```
    Estimate  Std. Error   t value     Pr(>|t|)
a  0.1296890 0.01977052  7.0404653 6.294826e-11
b  0.2299450 0.05860195  1.3491929 1.792946e-01
c -0.2604634 0.24463437 -0.8124437 4.178161e-01
```

CelebFaces:

```
     Estimate  Std. Error    t value     Pr(>|t|)
a  0.04402075 0.006472723   6.800963 7.779463e-11
b  2.42753078 0.450084264   5.393503 1.615915e-07
c -0.43706177 0.034317286 -12.735907 6.715243e-29
```

Por lo tanto, las curvas de aprendizaje esperadas quedarían así formalizadas:

$$A(n)_{Adult} = 0.870311 - 0.2299450 \cdot n^{-0.2604634} \tag{25}$$

$$A(n)_{CelebFaces} = 0.9559792 - 2.42753078 \cdot n^{-0.43706177} \tag{26}$$

◦ Estudio del valor de N:

Para poder estudiar el efecto del parámetro N correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera:

k1	**k2**	**k3**	**S**
10	4010	1000	5

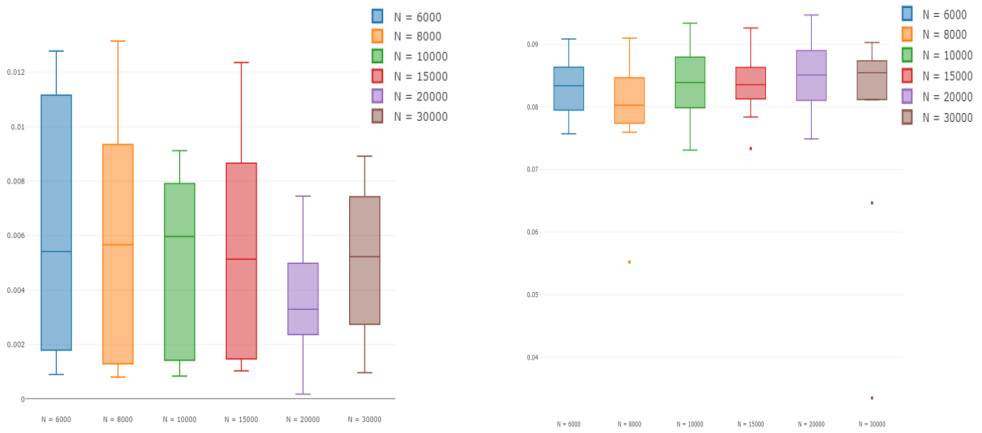
Tabla 2: Parámetros fijados del **Algoritmo LCP** para el estudio de N.

En este caso, y en el resto de trabajo, a la hora de medir el error entre la curva de aprendizaje esperada y la que se ha obtenido en cada caso, se ha calculado mediante el error absoluto medio (ecuación (6)).

En los gráficos de la Figura 8, se puede observar como cambia el error cometido en función de la N seleccionada. En este caso, se han tomado como posibles valores de N el conjunto de valores {6000, 8000, 10000, 15000, 20000, 30000}. Por otro lado, el error ha sido calculado en los puntos {6000:100:20000} para la base de datos *Adult*, y en los puntos {6000:1000:200000} para la base de datos *CelebFaces*. La razón por la que se han tomado estas dos listas diferentes, era para tener referencias del comportamiento del error al medirlo tanto a medio alcance como a largo alcance.

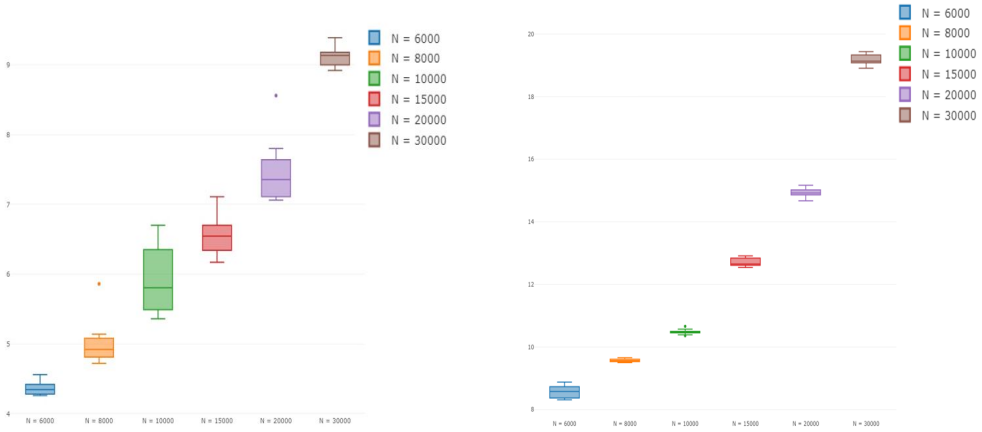
A su vez, en los gráficos de la Figura 9 se grafica el tiempo (en segundos) necesario para ejecutar el **Algoritmo LCP** con cada configuración de N.

Tanto en este caso, como en el resto de casos estudiados se han realizado un total de 10 pruebas independientes para medir tanto el error como el tiempo de cómputo necesario.



a: Adult b: CelebFaces

Figura 8: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 9: Tiempo(segundos) necesario para ejecutar el algoritmo con cada valor de **N**.

La clara conclusión que se saca de las gráficas, es que no se perciben mejoras significativas, en cuanto al error cometido, al aumentar el tamaño de la base de datos dada al algoritmo. En cambio, el tiempo de cómputo crece al aumentar N. Por lo que no parece que este aumento tenga grandes ventajas.

Aún y todo, al definir este parámetro, hay que tener en cuenta que su valor está en estrecha relación con los valores de k_2 y P.

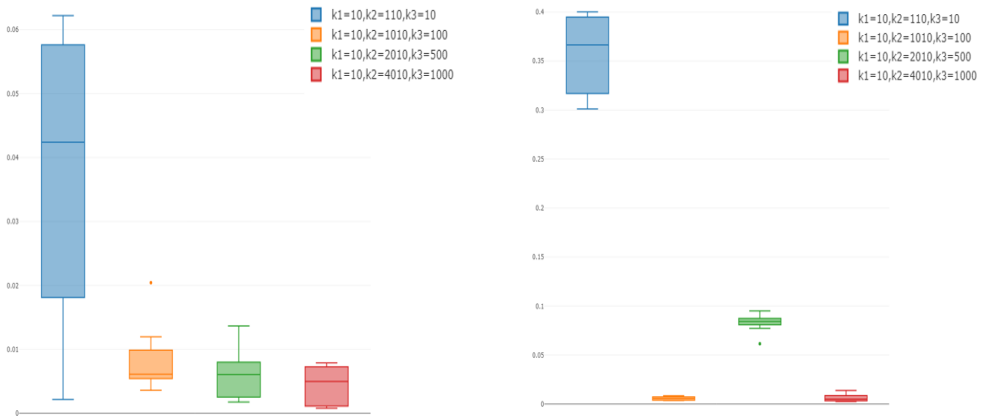
◦ Estudio de los valores de $k1, k2$ y $k3$:

Para poder estudiar el efecto de los parámetros k_i correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera:

N	**S**
8000	5

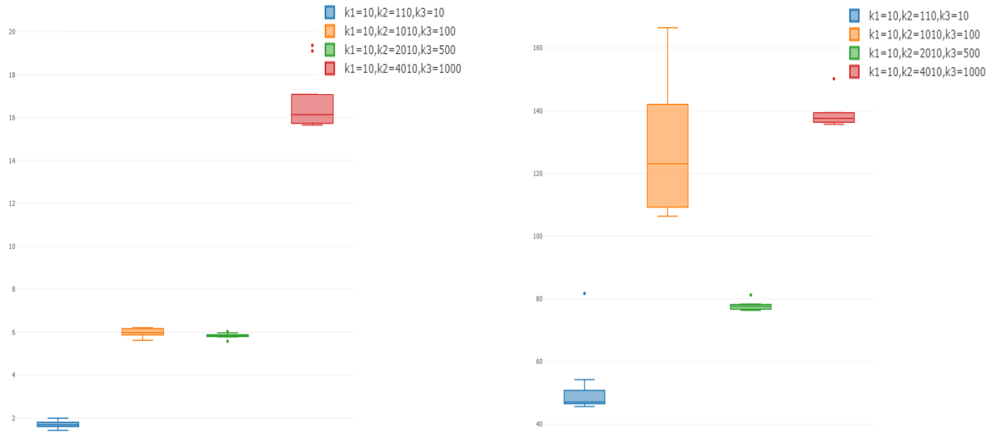
Tabla 3: Parámetros fijados del **Algoritmo LCP** para el estudio de k_i.

Las combinaciones de los valores de k_i seleccionados han sido: {[k_1=10, k_2=110, k_3=10], [k_1=10, k_2=1010, k_3=100], [k_1=10, k_2=2010, k_3=500], [k_1=10, k_2=4010, k_3=1000]}. La selección se ha hecho pensando en tener cubiertas las situaciones en las que el valor de k_3 es alto y bajo y que el número total de mediciones a realizar sea alto y bajo. Por su parte, el error se ha medido de la misma forma que en el caso anterior.



a: Adult b: CelebFaces

Figura 10: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 11: Tiempo(segundos) para ejecutar el algoritmo con distintos valores ki.

Observando, se percibe cómo en las gráficas con el error cometido (Figura 10), el resultado más visible es que el establecer un valor de k_2 bajo (boxplot azul) no es beneficioso, aunque en total se obtenga una nube de puntos con más puntos que en otros casos. A su vez, se destaca como en el caso de la base de datos *CelebFaces*, con la tercera configuración probada se obtienen unos resultados algo peores que con la tercera y cuarta configuración. Analizando por otro lado las gráficas que recogen el tiempo (en segundos) necesario para ejecutar el **Algoritmo LCP** en función de la configuración de los parámetros k_i, el resultado que más destaca es que cuando se establece un valor de k_2 alto el tiempo de cómputo crece considerablemente, mientras que el error cometido no se reduce en comparación con otras configuraciones.

En términos generales, tendiendo en cuenta el error cometido y el tiempo de cómputo necesario, las dos configuraciones intermedias (boxplot naranja y verde) quedan mejor posicionadas. Esto apunta a la necesidad de tener que mantener un equilibrio entre los parámetros k_i para tener los mejores resultados. También hay que tener en cuenta que el valor de k_2 está estrechamente ligado a los valores de N y P, debiendo tener en cuenta el valor del estos al definirlo.

◦ Estudio del valor de S:

Nuevamente, para poder estudiar el efecto del parámetro S correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera. En el caso de la base de datos *Adult* se ha usado:

k1	**k2**	**k3**	**N**
10	2010	500	8000

Tabla 4: Parámetros fijados del **Algoritmo LCP** para base de datos *Adult* en el estudio de S.

Y en la base de datos *CelebFaces* se ha usado:

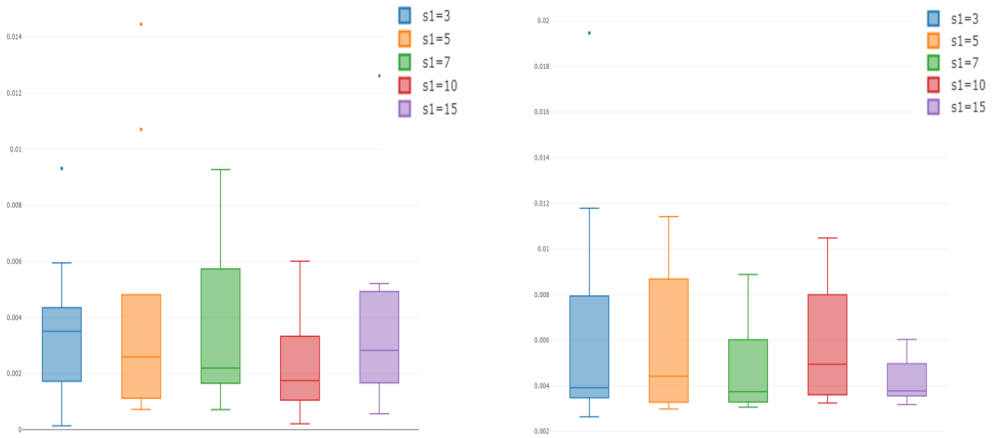
k1	**k2**	**k3**	**N**
10	1010	100	8000

Tabla 5: Parámetros fijados del **Algoritmo LCP** para base de datos *CelebFaces* en el estudio de S.

La razón de esta diferenciación se debe a los resultados obtenidos en el estudio de los valores k_i, donde en el caso de la base de datos *CelebFaces*, la combinación $\{k_1$=10, k_2=2010, k_3=500$\}$ mostraba un peor comportamiento respecto al error cometido.

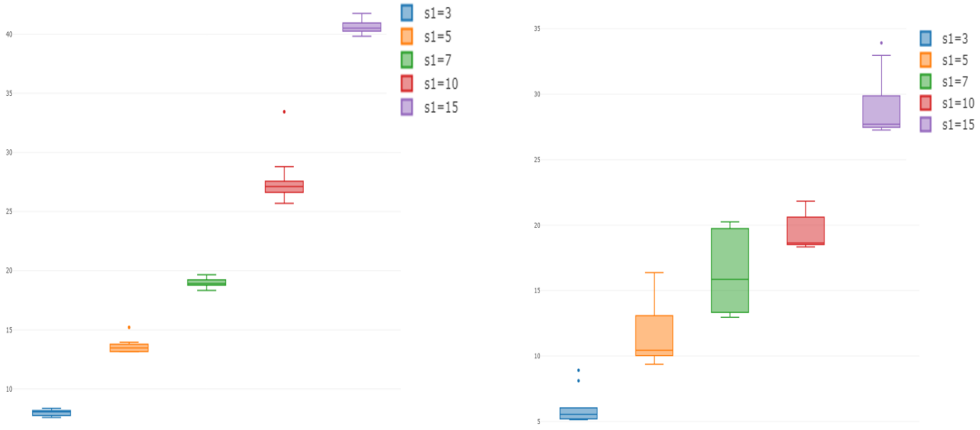
En los gráficos de la Figura 12 se puede ver cómo cambia el error cometido en función de la S seleccionada. El error se ha vuelto a calcular como en los casos anteriores. A su vez, los gráficos de la Figura 13 muestran el tiempo (en segundos) necesario para correr el **Algoritmo LCP** en función del valor de S.

Los resultados muestran claramente que, por un lado, no existen diferencias significativas, respecto al error, al usar una S más alta. Pero por otro lado, muestran que el tiempo de cómputo necesario crece notablemente al aumentar el valor de S. Por lo tanto, no se aprecia un beneficio en establecer valores altos de S.



a: Adult b: CelebFaces

Figura 12: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 13: Tiempo(segundos) para ejecutar el algoritmo con distintos valores **S**.

3.3.1.2. Modelos con RN

En el **Apéndice C** se encuentra el código en lenguaje `R` de la función programada para realizar las predicciones de la curva de aprendizaje con modelos de RN. Como en el caso anterior, en este también se han fijado los valores de los hiperparámetros de la RN. A continuación se muestra la configuración seleccionada:

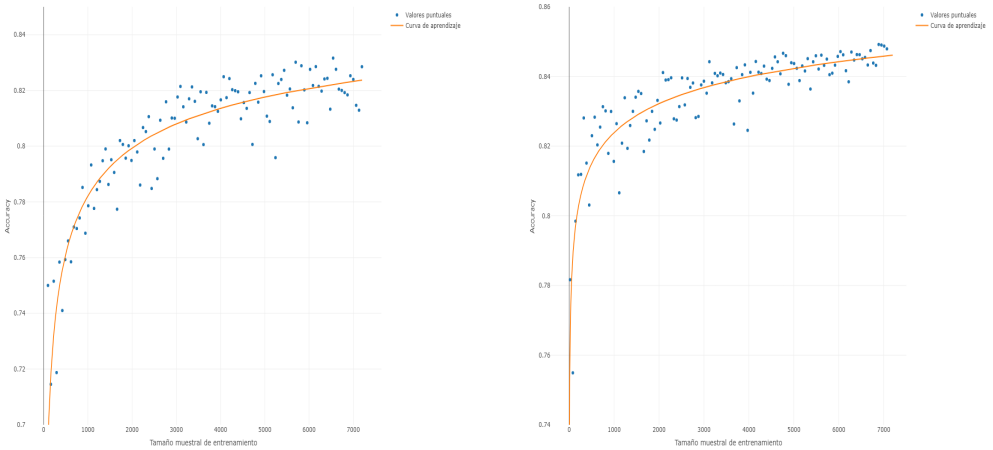
Número de capas ocultas	3
Función activación 1. capa oculta	relu
Número neuronas 1. capa oculta	100
Función activación 2. capa oculta	relu
Número neuronas 2. capa oculta	100
Función activación 3. capa oculta	relu
Número neuronas 3. capa oculta	100
Función activación capa salida	sigmoid
Número neuronas capa salida	1
optimizador	RMSprop
loss	MSE
Metrics	Accuracy
Epochs	25
Batch_Size	10

Tabla 6: Hiperparámetros del algoritmo RN fijados.

◦ Curvas de aprendizaje esperadas:

En este caso también se han calculado las que serán asumidas como las curvas de aprendizaje esperadas al aplicar el algoritmo RN con los hiperparámetros recogidos en la Tabla 6 a las bases de datos *Adult* y *CelebFaces*.

Para ello, se han seguido los mismos pasos que en el caso del SVM, pero tomando un rango de valores de X más reducido: [10,7000]. Esto se ha hecho para reducir los tiempos de cómputo y dando por hecho que la nube de puntos obtenida tendrá la suficiente cantidad de puntos como para estimar la curva de aprendizaje esperada con suficiente precisión.



a: Adult b: CelebFaces

Figura 14: Curvas de aprendizaje esperadas con RN.

A continuación, se muestran los resultados numéricos del proceso de mínimos cuadrados no lineales aplicado. Nuevamente, se destaca el hecho de que este método permite obtener la significatividad estadística de los parámetros estimados de la curva de aprendizaje.

Adult:

```
    Estimate Std. Error   t value     Pr(>|t|)
a  0.1186939 0.02529674  4.692061 7.969122e-06
b  0.6638338 0.22390899  2.964748 3.728407e-03
c -0.2752065 0.08483621 -3.243975 1.569058e-03
```

CelebFaces:

```
     Estimate Std. Error    t value     Pr(>|t|)
a  0.05209827 0.11885835  0.4383224 6.619821e-01
b  0.24737315 0.06117886  4.0434415 9.608108e-05
c -0.10000000 0.10405556 -0.9610251 3.385735e-01
```

Por lo tanto, las curvas de aprendizaje esperadas quedarían así formalizadas:

$$A(n)_{Adult} = 0.8813061 - 0.6638338 \cdot n^{-0.2752065} \tag{27}$$

$$A(n)_{CelebFaces} = 0.9479017 - 0.24737315 \cdot n^{-0.1} \tag{28}$$

◦ Estudio del valor de N:

Para poder estudiar el efecto del parámetro N correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera:

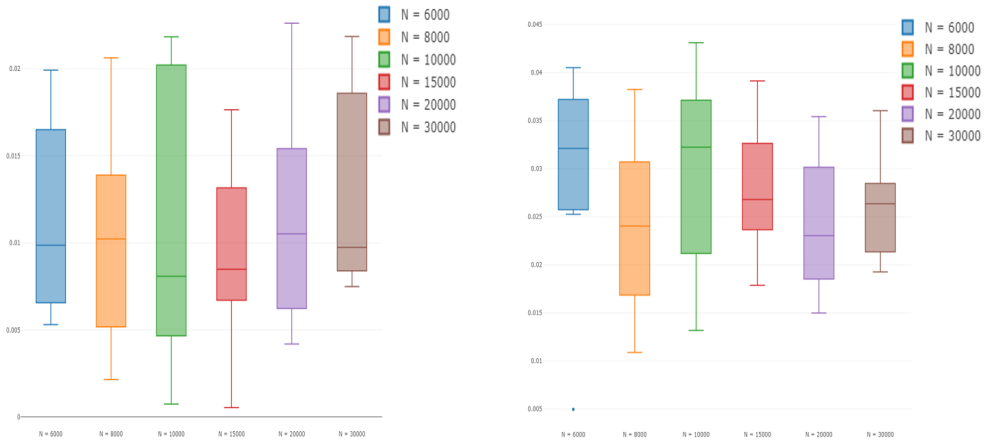
k1	**k2**	**k3**	**S**
10	2010	500	5

Tabla 7: Parámetros fijados del **Algoritmo LCP** para el estudio de N.

Con los modelos RN, se ha vuelto a medir el error haciendo uso del error absoluto medio (ecuación 6). Y se han vuelto a usar los puntos {6000:100:20000} para calcular el error en la base de datos *Adult*, y los puntos {6000:1000:200000} para la base de datos *CelebFaces*. A su vez, los valores de N con los que se ha trabajado han vuelto a ser: {6000, 8000, 10000, 15000, 20000, 30000}.

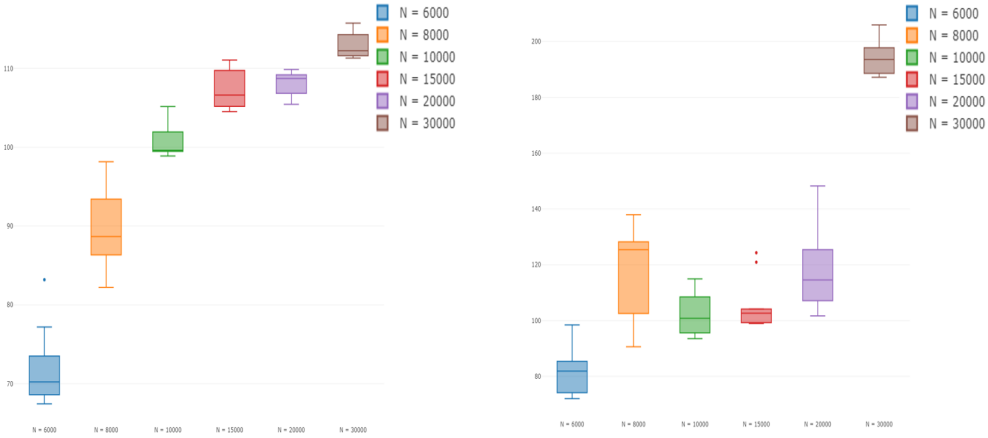
En los gráficos de la Figura 15 se puede observar como cambia el error cometido en función de la N seleccionada. Y en los gráficos de la Figura 16 se grafica el tiempo (en segundos) necesario para ejecutar el **Algoritmo LCP** con cada configuración de N.

Nuevamente, tanto en este caso, como en el resto de casos estudiados se han realizado un total de 10 pruebas independientes para medir tanto el error como el tiempo de cómputo necesario.



a: Adult b: CelebFaces

Figura 15: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 16: Tiempo(segundos) necesario para ejecutar el algoritmo con cada valor de **N**.

Otra vez, como en el caso del SVM, se obtiene la conclusión de que en lo que respecta al error cometido, no se perciben mejoras significativas al aumentar el tamaño de la base de datos dada al algoritmo. En cambio, en cuanto al tiempo de cómputo necesario corresponde, se ve como éste aumenta

al crecer el valor de N. Este último hecho se ve de forma más visible en la base de datos *Adult*.

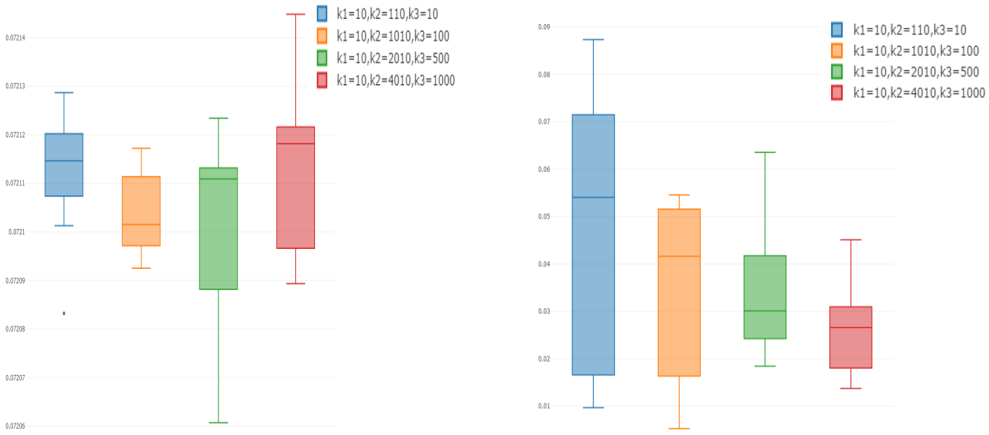
◦ Estudio de los valores de $k1, k2$ y $k3$:

Para poder estudiar el efecto de los parámetros k_i correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera:

N	**S**
8000	5

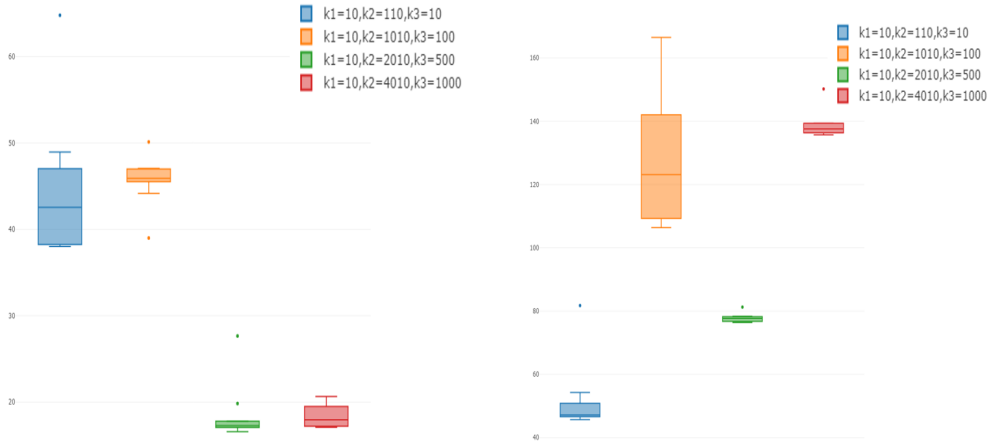
Tabla 8: Parámetros fijados del **Algoritmo LCP** para el estudio de k_i.

Las combinaciones de los valores de k_i seleccionados han sido las mismas que en el caso SVM: {[k_1=10, k_2=110, k_3=10], [k_1=10, k_2=1010, k_3=100], [k_1=10, k_2=2010, k_3=500], [k_1=10, k_2=4010, k_3=1000]}. Y el error se ha vuelto a medir de la misma manera.



a: Adult b: CelebFaces

Figura 17: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 18: Tiempo(segundos) para ejecutar el algoritmo con distintos valores ki.

Observando las gráficas se percibe que en este caso no es posible obtener resultados tan concluyentes como en situaciones anteriores. En cuanto al error, no se perciben relaciones significativas entre las distintas configuraciones de k_i y el error cometido, aunque en el caso de *CelebFaces* se percibe un ligero patrón descendente en las cuatro configuraciones probadas. Por lo que respecta al coste computacional, se ve como en el caso de la base de datos *Adult* han salido peor paradas las configuraciones que debían hacer un mayor número de evaluaciones. En cambio en la base de datos *CelebFaces* no se percibe un patrón claro.

◦ Estudio del valor de S:

Nuevamente, para poder estudiar el efecto del parámetro S correctamente, se han fijado el resto de parámetros en estudio de la siguiente manera. En el caso de la base de datos *Adult* se ha usado:

k1	**k2**	**k3**	**N**
10	2010	500	8000

Tabla 9: Parámetros fijados del **Algoritmo LCP** para base de datos *Adult* en el estudio de S.

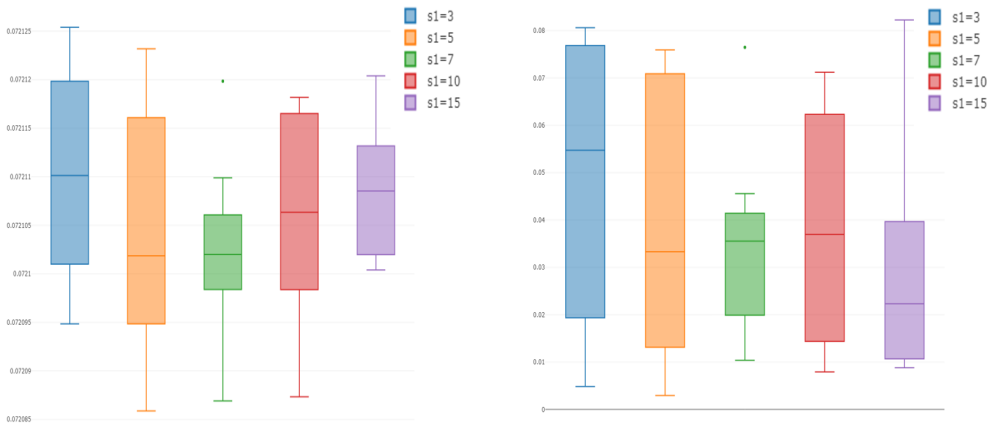
Y en la base de datos *CelebFaces* se ha usado:

k1	**k2**	**k3**	**N**
10	110	10	8000

Tabla 10: Parámetros fijados del **Algoritmo LCP** para base de datos *CelebFaces* en el estudio de S.

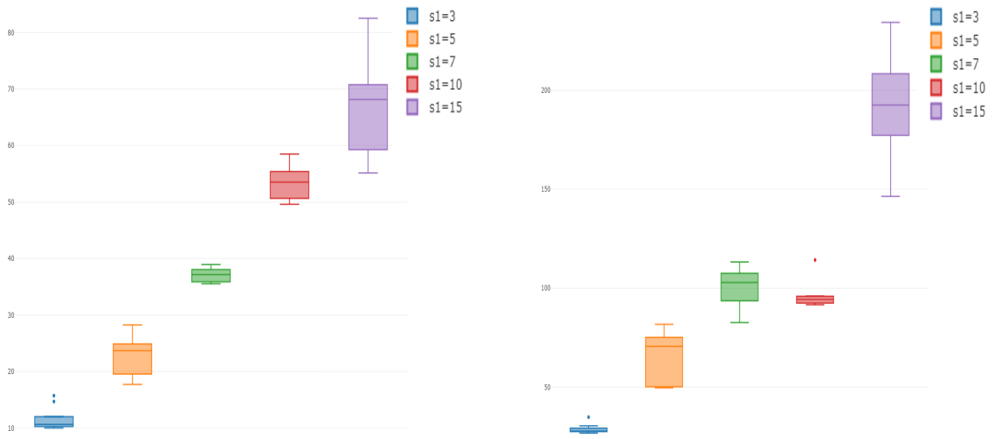
La razón de esta diferenciación se debe a los resultados obtenidos en el estudio de los valores k_i. Y se han seleccionado las opciones que han tenido un menor coste computacional. En ambos casos, las configuraciones seleccionadas tenían un menor coste computacional y un error asociado similar al resto de opciones.

En los gráficos de la Figura 19 se puede ver como cambia el error cometido en función de la S seleccionada. El error se ha vuelto a calcular como en los casos anteriores. A su vez, los gráficos de la Figura 20 muestran el tiempo (en segundos) necesario para correr el **Algoritmo LCP** en función del valor de S. Los resultados muestran claramente que, por un lado, no existen diferencias significativas, respecto al error, en usar una S más alta. Pero por otro lado, muestran que el tiempo de cómputo necesario crece notablemente al aumentar el valor de S.



a: Adult b: CelebFaces

Figura 19: Error Absoluto Medio entre la curva esperada y la estimación.



a: Adult b: CelebFaces

Figura 20: Tiempo(segundos) para ejecutar el algoritmo con distintos valores **S**.

3.3.1.3. Conclusiones

Tras analizar los distintos casos expuestos, las conclusiones sobre la configuración de los parámetros del **Algoritmo LCP** son las siguientes. En primer lugar, en lo que al tamaño N corresponde, se ha visto que no compensa tomar valores muy grandes, pues mientras que el coste computacional asociado aumenta, el error cometido se mantiene relativamente estable. En cuanto a los valores k_i, no se percibe que puedan sacarse conclusiones generales, pero como recomendación se apunta a no tomar una configuración que tenga que realizar excesivas evaluaciones ni que tenga un valor k_2 muy alto. Por último, con S tampoco se ha visto que tomar valores altos tenga un especial beneficio. Por lo que se recomienda un valor bajo pero que introduzca algo de variabilidad, como por ejemplo S=3. A su vez, hay que tener en cuenta que los valores N, P y k_2 son dependientes, y que el valor de uno condiciona al resto.

4. Experimentación

El objetivo de la fase experimental es verificar que el uso de la estimación de las curvas de aprendizaje para predecir el rendimiento de un modelo de aprendizaje automático, puede acelerar la optimización bayesiana frente al uso de la validación cruzada. Por lo tanto, la experimentación se ha centrado en comparar los resultados y el coste computacional asociado con cada uno de los métodos.

Tal y como se ha visto a lo largo del trabajo, al usar la optimización bayesiana en la práctica, suele utilizarse la validación cruzada para medir el rendimiento de un modelo. Suponiendo un caso concreto en el que se evalúa el rendimiento mediante validación cruzada, haciendo uso de grupos de G_1 individuos para el entrenamiento y grupos de G_2 individuos para la validación (donde $G_1 = c \cdot G_2 \colon c \in \mathbf{N}$); el rendimiento medio obtenido como resultado de las validaciones se asume como una buena estimación del rendimiento que podemos esperar al entrenar el modelo con G_1 datos. Es por ello que, en el diseño de la fase experimental, se ha decidido tomar como la solución de referencia la obtenida usando el método de validación cruzada, y se compara con el uso del método de curvas de aprendizaje. Por lo tanto, el objetivo del experimento es ver si la calidad de los resultados de la optimización bayesina obtenidos con curvas de aprendizaje, son lo suficientemente buenos y si representan una ventaja real en la reducción del coste computacional del proceso de optimización.

En la experimentación se han vuelto a usar los mismos algoritmos de aprendizaje automático utilizados en el subapartado 3.3.1, es decir, el *support vector machine* (SVM) y la red neuronal del tipo perceptrón multicapa (RN). Las razones de usar estos algoritmos son las mismas que las presentadas en dicho apartado. Se ha querido experimentar con un algoritmo del área del aprendizaje automático clásico y otro del área del *deep learning*. El motivo de haber seleccionado el SVM es por los buenos resultados que muestra y por el elevado coste computacional que tiene al trabajar con grandes bases de datos. Por otro lado, se ha seleccionado el perceptrón multicapa por ser un modelo básico del *deep learning* y, a su vez, de los más utilizados.

Para la experimentación se ha trabajado con los datos públicos de la

Agencia Espacial Europea (ESA) sobre la misión GAIA[6]. GAIA es una sonda espacial de la ESA que fue lanzada el 19 de diciembre de 2013, y es usada dentro de la misión homónima donde proporciona astrometría, fotometría y espectroscopia de más de 1000 millones de estrellas en la Vía Láctea. Con el objetivo de trazar un mapa tridimensional de nuestra Galaxia, la Vía Láctea, en el proceso que revela la composición, formación y evolución de la Galaxia. En concreto, se ha trabajado con una base de datos con un total de 32 variables y 1310718 observaciones. Para poder entrenar un clasificador binario, se ha binomizado la variable que indica el número de malas observaciones astrométricas realizadas en escaneo largo, convirtiéndola en una variable que informa si existen o no malas observaciones.

En toda la experimentación, se ha usado la función de adquisición *expected improvement* (EI), puesto que en la literatura se destaca que es capaz de mantener un buen equilibrio entre la exploración y la explotación del espacio muestral. A su vez, en la optimización bayesiana, se han tomado 8 mediciones iniciales desde donde comenzar las iteraciones de optimización en el caso del SVM y 7 en el de RN. El número de mediciones iniciales no ha sido seleccionado aleatoriamente, sino que se ha tenido en cuenta que debe ser mayor que el número de parámetros a optimizar, y que cuantas más evaluaciones se realizan más aumenta el tiempo de cómputo necesario. Por último, se han realizado 5 iteraciones de optimización bayesiana, donde en cada una se han tomado las dos posibles mejores combinaciones de hiperparámetros respecto a EI.

La combinación de parámetros del **Algoritmo LCP** seleccionada para experimentar ha sido la siguiente:

N	**k1**	**k2**	**k3**	**S**	**l**
2010	10	1010	250	3	10000

Tabla 11: Parámetros fijados del **Algoritmo LCP** para la experimentación.

Esta selección de valores se ha realizado teniendo en cuenta las conclusiones obtenidas en el subapartado 3.3.1. Esto supone que, para evaluar el rendimiento de una configuración de hiperparámetros se sigue el siguiente

[6] `https://gea.esac.esa.int/archive/`

proceso: Partiendo de una muestra aleatoria de **N=2010** individuos, en primer lugar, se entrena el modelo de manera independiente utilizando como tamaños de conjunto de entrenamiento los valores {10,260,510,760,1010}, utilizando para la validación todos aquellos datos no utilizados para el entrenamiento y realizando la división de forma aleatoria. Para cada uno de esos valores, se entrena y evalúa el modelo un total de **S=3** veces, tomando la media como rendimiento esperado. Con los resultados obtenidos, se estima la curva de aprendizaje; y con dicha curva se predice cual será el rendimiento esperado al entrenar el modelo con **l=10000** individuos.

A su vez, para tener una referencia con la que comparar, se ha aplicado la optimización bayesiana con el método de la validación cruzada. Para ello se han tomado muestras aleatorias de 15000 individuos y se ha aplicado un *3-fold cross validation*. Lo que implica que el resultado de la optimización obtenido, se asume como una solución de referencia en caso de hacer uso de 10000 datos de entrenamiento.

En un inicio, el objetivo de la experimentación era trabajar a una escala más grande, pero los altos costes computacionales de algunos de los procesos no lo han permitido.

4.1. Modelos con SVM

Antes de nada, se han definido los hiperparámetros de SVM a optimizar junto con su espacio de búsqueda. Y a su vez, se han establecido que hiperparámetros se han fijado previamente.

Hiperparámetro	**Posibles valores**
Degree	{1,2,3,4,5}
Gamma	$[2^{-15}, 2^{3}]$
Coef0	[-1,1]
Cost	{1,2}
Tolerance	$[10^{-5}, 10^{-1}]$
Epsilon	[0.001,0.5]
Shirinking	{True,False}

Tabla 12: Hiperparámetros de SVM a optimizar.

Hiperparámetro	Valor fijado
Type	'C-classification'
Kernel	'polynomial'

Tabla 13: Hiperparámetros de SVM fijados.

En este caso, con el objetivo de acelerar los cálculos, el problema se ha paralelizado. Dicha paralelización ha consistido en repartir los cálculos necesarios entre 8 clusters de procesamiento diferentes. Esto ha sido posible gracias a la librería `doParallel` de `R`, y la posibilidad que ofrece la optimización bayesiana de paralelizar los cálculos.

Uso de la validación cruzada para la estimación de rendimiento

Debido al alto coste computacional asociado a este proceso, únicamente se han llevado a cabo un total de 5 ejecuciones independientes. En la Tabla 14, se muestra un resumen estadístico del coste temporal (en segundos).

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
66143	67334	67490	69936	72426	76286

Tabla 14: Coste computacional de la optimización bayesiana con VC.

Los resultados muestran que son necesarias más de 19 horas de media para completar la optimización bayesiana utilizando la validación cruzada para medir el rendimiento. Este elevado coste computacional es debido a que el Algoritmo SVM tiene una complejidad del orden $O(n^3)$ [7] [35].

En la Tabla 15, se muestran los resultados obtenidos en cada ejecución, donde podemos ver la combinación de hiperparámetros seleccionada y el rendimiento (Accuracy) asociado.

[7] `https://www.thekerneltrip.com/machine/learning/computational-complexity-learning-algorithms/`

Ejec	D	G	Coef	Cost	Tol	Eps	Shiri	Accuracy
1	1	1.364	-0.938	2	0.035	0.500	FALSE	0.6723105
2	1	6.890	0.289	2	0.084	0.0826	TRUE	0.6727104
3	1	4.569	0.407	1	0.089	0.027	FALSE	0.6723772
4	1	7.608	0.740	1	0.036	0.001	TRUE	0.6725100
5	1	4.912	0.701	1	0.072	0.415	FALSE	0.6727762

Tabla 15: Resultados de las ejecuciones con SVM y validación cruzada.

Uso de curvas de aprendizaje para la estimación de rendimiento

En el **Apéndice D** se encuentra el código de `R` con el que se ha realizado la optimización bayesiana para el algoritmo SVM con la predicción de la curva de aprendizaje. En este caso se han realizado un total de 10 ejecuciones independientes, y a continuación se muestra un resumen estadístico del coste temporal (en segundos) de los mismos.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
177.0	248.6	284.7	299.7	344.7	415.4

Tabla 16: Coste computacional de la optimización bayesiana con CA.

En los resultados de la Tabla 16, se puede observar como son necesarios en torno a 5 minutos de media para completar la optimización bayesiana haciendo uso del **Algoritmo LCP** para medir los rendimientos. A continuación, se muestran los resultados obtenidos en cada ejecución, donde podemos ver la combinación de hiperparámetros seleccionada y el rendimiento (Accuracy) asociado.

Ejec	D	G	Coef	Cost	Tol	Eps	Shiri	Accuracy
1	1	7.457	0.388	2	0.085	0.188	FALSE	0.6766098
2	1	0.062	0.233	2	0.011	0.500	TRUE	0.7070098
3	1	7.784	0.473	1	0.084	0.001	FALSE	0.6951548
4	1	6.721	-0.782	2	0.050	0.093	TRUE	0.7378667
5	1	0.502	0.043	1	0.001	0.128	TRUE	0.7105807
6	1	0.984	0.009	1	0.009	0.500	TRUE	0.7111867
7	1	8.000	0.956	1	0.095	0.038	FALSE	0.7138451
8	1	6.051	0.306	1	0.069	0.500	FALSE	0.6857512

Ejec	D	G	Coef	Cost	Tol	Eps	Shiri	Accuracy
9	1	4.068	0.434	1	0.081	0.001	FALSE	0.7188314
10	1	6.739	0.298	2	0.082	0.043	TRUE	0.7235494

Tabla 17: Resultados de las ejecuciones con SVM y curvas de aprendizaje.

En primer lugar, en cuanto al tiempo de cómputo se refiere, es destacable que entre un método y otro existe una gran diferencia. Mientras que con el primer método se han necesitado más 19 horas de media, el segundo método no ha necesitado más que 5 minutos de media para completar el proceso. Esto supone una reducción de más del 99 %, por lo que en cuanto al tiempo de cómputo se refiere, se han obtenido resultados más que favorables en favor del uso de método de curvas de aprendizaje.

En segundo lugar, en lo que respecta a la configuración de hiperparámetros obtenida con cada método, se ha querido ver si existe una clara diferenciación entre las soluciones. En caso de que las soluciones logradas con la validación cruzada, las cuales se asumen como soluciones de referencia, se agrupasen de manera conjunta diferenciándose claramente de las soluciones obtenidas por curvas de aprendizaje; podría indicar la existencia de uno o varios máximos que el segundo método no está siendo capaz de identificar. Esto último sería un claro error del método basado en curvas de aprendizaje, lo que pondría en cuestión su validez. Puesto que a simple vista es complicado aceptar o rechazar una clara diferenciación entre los resultados, se ha recurrido a la técnica del análisis de las componentes principales. Tal y como sabemos de la teoría, el análisis de componentes principales puede ser usado para la determinación de datos homogéneos o *clusters* [36]. Analizando la representación de las diferentes soluciones encontradas respecto a las dos primeras componentes principales (Figura 21), no se percibe que las soluciones del método de la validación cruzada se agrupen de manera diferenciada. A su vez, se ve que en cuatro casos ambos métodos han obtenido resultados similares.

Por último, en cuanto a los *accuracy* devueltos en el segundo caso por la optimización bayesiana, los cuales han sido estimados por curvas de aprendizaje, se puede observar variabilidad entre los casos. Esto se debe a que la calidad de las proyecciones de las curvas de aprendizaje realizadas no son siempre óptimas, y pueden sobrevalorar o infravalorar el rendimiento futuro

de una combinación dada. Pero más allá de la variabilidad existente, la cual puede ser causada por la variación de las bases de datos utilizadas en cada caso, o simplemente por la naturaleza probabilística del método, tal y como se ve en la Figura 22 no hay existencia de *outliers* en los resultados obtenidos. Esto último, demuestra que el uso de este método da ciertas garantías de robustez frente a la variabilidad.

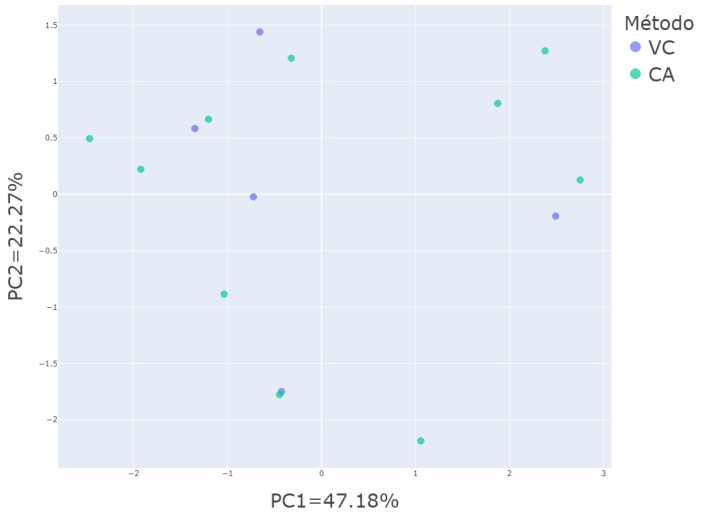


Figura 21: Nube de puntos de las dos primeras componentes principales.

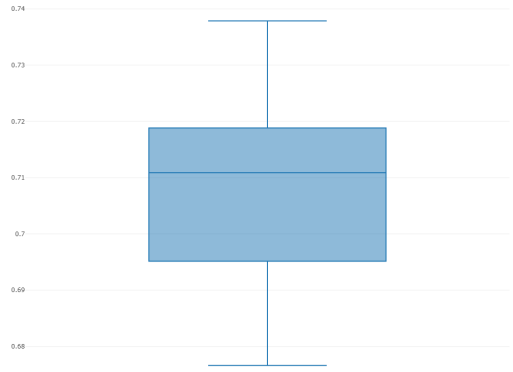


Figura 22: Boxplot de rendimientos obtenidos con CA.

4.2. Modelos con RN

Tal y como se ha realizado con al algoritmo anterior, antes de nada, se han definido los hiperparámetros de RN a optimizar junto con su espacio de búsqueda y los hiperparámetros que se han fijado previamente.

Hiperparámetro	Posibles valores
Número de neuronas en 1. capa	{1,2,...,250}
Número de neuronas en 2. capa	{1,2,...,250}
Número de neuronas en 3. capa	{1,2,...,250}
Learning Rate	[0,1]
RHO	[0,1]
Decay	[0,1]

Tabla 18: Hiperparámetros de RN a optimizar.

Hiperparámetro	Valores fijados
Número de capas ocultas	3
Función activación capas ocultas	relu
Función activación capa salida	sigmoid
Número neuronas capa salida	1
optimizador	RMSprop
loss	MSE
Metrics	Accuracy
Epochs	25
Batch_Size	10

Tabla 19: Hiperparámetros de RN fijados.

En este caso, el problema no ha sido posible paralelizarlo por la incompatibilidad de las librerías encargadas de la paralelización y de las redes neuronales.

Uso de la validación cruzada para la estimación de rendimiento

En este experimento, el coste computacional ha permitido realizar 10 ejecuciones independientes. A continuación, se muestra un resumen estadístico del coste temporal (en segundos).

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
1491	1558	1627	1650	1715	1922

Tabla 20: Coste computacional de la optimización bayesiana con VC.

Los resultados muestran como son necesarios 27,5 minutos de media para completar la optimización bayesiana utilizando la validación cruzada para medir el rendimiento. Y a continuación, se muestran los resultados obtenidos en cada ejecución, donde podemos ver la combinación de hiperparámetros seleccionada y el rendimiento (Accuracy) asociado.

Ejec	L1	L2	L3	LR	RHO	D	Accuracy
1	1	1	250	0.000	0.342	0.000	0.6223175
2	1	250	1	0.070	0.597	0.402	0.6223176
3	239	239	1	0.423	0.000	0.888	0.6223174
4	43	23	63	0.590	0.810	0.814	0.6223174
5	250	250	1	0.934	0.924	0.522	0.6223178
6	169	186	58	0.516	0.768	0.283	0.6223174
7	38	122	33	0.804	0.216	0.038	0.6223175
8	9	110	28	0.019	0.636	0.747	0.6472484
9	87	32	201	0.013	0.276	0.610	0.6596457
10	1	250	1	0.871	0.165	0.288	0.6223171

Tabla 21: Resultados de las ejecuciones con RN y validación cruzada.

Uso de curvas de aprendizaje para la estimación de rendimiento

En el **Apéndice E** se encuentra el código de `R` con el que se ha realizado la optimización bayesiana para el algoritmo RN con la predicción de la curva de aprendizaje. En este caso se han realizado un total de 10 ejecuciones independientes, y a continuación se muestra un resumen estadístico del coste temporal (en segundos) de los mismos.

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
645	714	728	791	772	1197

Tabla 22: Coste computacional de la optimización bayesiana con CA.

En los resultados de la Tabla 22, se puede observar que son necesarios en torno a 13 minutos de media para completar la optimización bayesiana haciendo uso del **Algoritmo LCP** para medir los rendimientos. A continuación, se muestran los resultados obtenidos en cada ejecución, donde podemos ver la combinación de hiperparámetros seleccionada y el rendimiento (*Accuracy*) asociado.

Ejec	**L1**	**L2**	**L3**	**LR**	**RHO**	**D**	**Accuracy**
1	1	250	1	0.659	0.546	1.000	0.6223881
2	51	35	2	0.419	0.567	0.252	0.6614665
3	250	1	151	0.109	0.535	0.222	0.6614665
4	90	28	30	0.001	0.790	0.305	0.6354190
5	119	215	155	0.514	0.027	0.188	0.6687200
6	250	1	1	0.000	1.000	0.367	0.6396986
7	1	250	1	0.629	0.601	0.688	0.6223881
8	113	9	109	0.677	0.857	0.968	0.6325087
9	1	250	1	0.678	0.549	0.829	0.6223881
10	1	1	250	0.750	0.806	0.996	0.6223881

Tabla 23: Resultados de las ejecuciones con RN y curvas de aprendizaje.

En primer lugar, en cuanto al tiempo de cómputo se refiere, es visible que la diferencia entre un método y otro ya no es tan elevada. Aún y todo, mientras que con el primer método se han necesitado 27.5 minutos media, el segundo método ha necesitado algo más de 13 minutos de media para completar el proceso. Esto supone una reducción de más del 50 %, por lo que, aunque la diferencia no es tan grande como en el primer caso, se han vuelto a obtener resultados más que favorables en favor del uso de método de curvas de aprendizaje.

En segundo lugar, en lo que respecta a la configuración de hiperparámetros obtenida con cada método, se ha recurrido nuevamente a la técnica del análisis de las componentes principales. Analizando la representación de las diferentes soluciones encontradas respecto a las dos primeras componentes principales (Figura 23), no se percibe la existencia de una clara diferenciación entre las soluciones de los dos métodos. A su vez, se vuelve a percibir similitud entre varias soluciones encontradas con cada método

Por último, en cuanto a los *accuracy* devueltos por la optimización bayesiana, los cuales han sido estimados por curvas de aprendizaje, se vuelve a apreciar cierta variabilidad entre los casos, aunque menor que en el experimento anterior. Pero la ausencia de *outliers* entre dichos valores, tal y como se ve en la Figura 24, demuestra nuevamente que el uso de curvas de aprendizaje como método de estimación de rendimientos da ciertas garantías de robustez frente a la variabilidad.

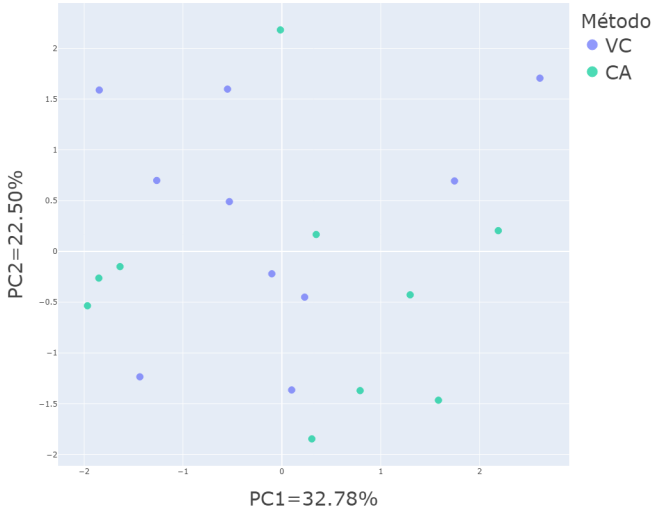


Figura 23: Nube de puntos de las dos primeras componentes principales.

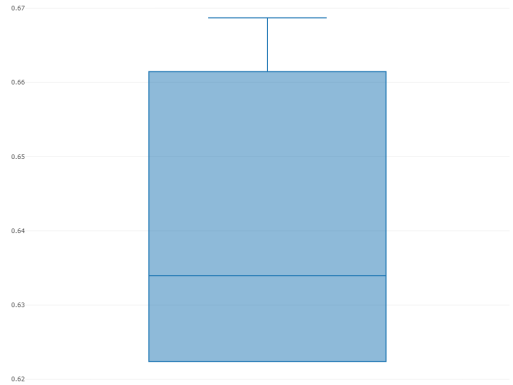


Figura 24: Boxplot de rendimientos obtenidos con CA.

5. Conclusiones y trabajos futuros

En este trabajo de fin de máster, tras analizar el estado del arte en lo que respecta a las curvas de aprendizaje y la optimización de hiperparámetros, se ha planteado la hipótesis de que el uso de las curvas de aprendizaje como método de predecir el rendimiento de un modelo de aprendizaje automático, podría ser de utilidad a la hora de reducir el coste computacional de la optimización bayesiana. Esta hipótesis, se sostiene sobre la intuición de que dicho método debiera ser notablemente menos costoso que la validación cruzada, técnica utilizada comúnmente en la optimización bayesiana para medir el rendimiento de un modelo dado.

Tras estudiar las bases teóricas de las curvas de aprendizaje y la optimización bayesiana, se ha diseñado y analizado la mejor forma de usar un método capaz de predecir el rendimiento de un modelo en base a la estimación de las curvas de aprendizaje. En la fase experimental del trabajo se ha comparado dicho método con el uso de la validación cruzada. Los resultados obtenidos, han venido a confirmar la conjetura planteada.

En los dos experimentos realizados, el primero con modelos SVM y el segundo con modelos RN, se han reducido el tiempo medio de ejecución de la optimización bayesiana en un 99 % y en un 50 % respectivamente. Por lo que los resultados son más que favorables en favor del uso de las curvas de aprendizaje. A su vez, hay que tener en cuenta que en este caso se ha experimentado en una escala reducida en cuanto a la cantidad de datos se refiere; y se espera que estas diferencias aumenten en escalas de trabajo mucho más grandes. En lo que respecta a las configuraciones de hiperparámetros devueltas como solución, no se han detectado diferencias significativas entre los resultados obtenidos con ambos métodos, y esto último también va en favor del método de curvas de aprendizaje.

Los beneficios que puede acarrear la adopción de esta técnica en la práctica son considerables, principalmente, en la optimización de hiperparámetros en entornos de *Big Data*. Ahí se espera que las diferencias de tiempo de cómputo entre ambos métodos se amplíe en favor, nuevamente, de las curvas de aprendizaje.

No obstante, aunque los resultados sean favorables e inviten al optimis-

mo, todavía queda trabajo por hacer en esta área de investigación, y con ello se establecen futuros trabajos y nuevas vías de investigación.

En primer lugar, en lo que respecta a los estudios más básicos de esta área, se ve la necesidad de profundizar en el estudio de las curvas de aprendizaje, con el objetivo de conseguir métodos más precisos para predecir las curvas de aprendizaje. Estos avances pueden venir, entre otros, desde el aspecto matemático presentando nuevas formas de modelizar las curvas, o desde el aspecto algorítmico mejorando los algoritmos aquí expuestos, como por ejemplo, optimizando el algoritmo encargado de realizar la regresión de mínimos cuadrados no lineales(`nls`).

En segundo lugar, se percibe la necesidad de ampliar tanto el estudio realizado en el subapartado 3.3.1, como en la parte experimental del trabajo, a nuevos tipos de algoritmos y nuevas bases de datos, de cara a seguir fortaleciendo la hipótesis planteada. A su vez, se ve necesario ampliar el problema más allá de la clasificación binaria, a problemas de clasificación múltiple y regresión. Otro de los asuntos que no se ha abordado en el trabajo y que sería conveniente analizarlo, es la frontera de rentabilidad, es decir, a partir de que tamaño de bases de datos se convierte rentable hacer uso del método propuesto.

Por último, se puede abrir una nueva vía de investigación combinando el método de curvas de aprendizaje con otros métodos de optimización de hiperparámetros, como por ejemplo los métodos evolutivos, los cuales permiten un encaje perfecto con la técnica propuesta.

Para finalizar, se quiere analizar el aspecto ético del trabajo aquí expuesto. Normalmente, para hacer la selección de hiperparámetros de forma manual, hay que realizar un análisis exhaustivo, donde el conocimiento de la función de cada uno de los hiperparámetros es fundamental. Esto tiene implicaciones directas sobre la acción y supervisión humanas, dado que se pueden crear modelos matemáticamente óptimos pero que por la selección de hiperparámetros estén generando patrones indeseados, como por ejemplo sesgos injustos contrarios a la diversidad, no discriminación y equidad.

La única manera de evitar este tipo de problemas es que el usuario sea capaz de analizar en profundidad el funcionamiento del algoritmo obtenido. Al fin y

al cabo, el objetivo reside en obtener resultados óptimos en el menor tiempo posible, pero esto no puede eximir al usuario del deber de conocer y ser capaz de analizar la herramienta que está creando. Por lo que, de cara a recordar al usuario final los puntos analizados aquí, se valora positivamente que de cara a implementar este método, el algoritmo devuelva una advertencia de texto al programador con el concepto de que, dependiendo del área de aplicación, los modelos matemáticamente más óptimos no son siempre los más adecuados, y que siempre se debe hacer un análisis al respecto para evitar sesgos indeseados.

Bibliografía

[1] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei y S.-H. Deng, "Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb", *Journal of Electronic Science and Technology*, vol. 17, n.º 1, págs. 26-40, 2019, ISSN: 1674-862X. DOI: https://doi.org/10.11989/JEST.1674-862X.80904120.

[2] J. Bergstra e Y. Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, vol. 13, n.º 10, págs. 281-305, 2012. dirección: http://jmlr.org/papers/v13/bergstra12a.html.

[3] P. Brazdil, C. Giraud-Carrier, C. Soares y R. Vilalta, *Metalearning - Applications to Data Mining*. 2009, ISBN: 978-3-540-73262-4. DOI: 10.1007/978-3-540-73263-1.

[4] D. Jones, "A Taxonomy of Global Optimization Methods Based on Response Surfaces", *J. of Global Optimization*, vol. 21, págs. 345-383, dic. de 2001. DOI: 10.1023/A:1012771025575.

[5] R. L. Figueroa, Q. Zeng-Treitler, S. Kandula y L. H. Ngo, "Predicting sample size required for classification performance", *BMC Medical Informatics and Decision Making*, vol. 12, n.º 8, 2012.

[6] S. Mukherjee, P. Tamayo, S. Rogers, R. Rifkin, A. Engle, C. Campbell, T. Golub y J. Mesirov, "Estimating Dataset Size Requirements for Classifying DNA Microarray Data", *Journal of computational biology : a journal of computational molecular cell biology*, vol. 10, págs. 119-42, feb. de 2003. DOI: 10.1089/106652703321825928.

[7] A. N. Richter y T. M. Khoshgoftaar, "Learning Curve Estimation with Large Imbalanced Datasets", en *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, 2019, págs. 763-768. DOI: 10.1109/ICMLA.2019.00135.

[8] S. K. Lwanga, S. Lemeshow y W. H. Organization, *Sample size determination in health studies : a practical manual*. Geneva: World Health Organization, 1991.

[9] B. Settles, "Active Learning Literature Survey", University of Wisconsin–Madison, Computer Sciences Technical Report 1648, 2009.

[10] F. Provost, D. Jensen y T. Oates, "Efficient Progressive Sampling", KDD '99, págs. 23-32, 1999. DOI: 10.1145/312129.312188.

[11] C. Cortes, L. D. Jackel, S. Solla, V. Vapnik y J. Denker, "Learning Curves: Asymptotic Values and Rate of Convergence", en *Advances in Neural Information Processing Systems*, J. Cowan, G. Tesauro y J. Alspector, eds., vol. 6, Morgan-Kaufmann, 1994. dirección: https://proceedings.neurips.cc/paper/1993/file/1aa48fc4880bb0c9b8a3bf979d3b917e-Paper.pdf.

[12] Markellos, Black y Moran, "A Grid Search for Families of Periodic Orbits in the Restricted Problem of Three Bodies", *Celestial Mechanics*, vol. 9, págs. 507-512, 1974. DOI: 10.1007/BF01329331.

[13] Y. Bao y Z. Liu, "A Fast Grid Search Method in Support Vector Regression Forecasting Time Series", en *Intelligent Data Engineering and Automated Learning – IDEAL 2006*, E. Corchado, H. Yin, V. Botti y C. Fyfe, eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, págs. 504-511, ISBN: 978-3-540-45487-8.

[14] Dirección: https://github.com/jaak-s/nips2014-survey.

[15] J. Bergstra e Y. Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, vol. 13, n.º 10, págs. 281-305, 2012. dirección: http://jmlr.org/papers/v13/bergstra12a.html.

[16] J. Snoek, H. Larochelle y R. P. Adams, *Practical Bayesian Optimization of Machine Learning Algorithms*, 2012. arXiv: 1206.2944 [stat.ML].

[17] E. Brochu, V. M. Cora y N. de Freitas, *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, 2010. arXiv: 1012.2599 [cs.LG].

[18] J. Joyce, "Bayes' Theorem", en *The Stanford Encyclopedia of Philosophy*, E. N. Zalta, ed., Spring 2019, Metaphysics Research Lab, Stanford University, 2019.

[19] M. Nomura, *Simple and Scalable Parallelized Bayesian Optimization*, 2020. arXiv: 2006.13600 [stat.ML].

[20] E. Carmona y S. Fernández, *Fundamentos de la Computación Evolutiva*. ene. de 2020, ISBN: 978-84-267-2755-8.

[21] J.-Y. Kim y S.-B. Cho, "Evolutionary Optimization of Hyperparameters in Deep Learning Models", en *2019 IEEE Congress on Evolutionary Computation (CEC)*, 2019, págs. 831-837. DOI: 10.1109/CEC.2019.8790354.

[22] J. Waring, C. Lindvall y R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare", *Artificial Intelligence in Medicine*, vol. 104, pág. 101 822, 2020, ISSN: 0933-3657. DOI: https://doi.org/10.1016/j.artmed.2020.101822.

[23] X. Zeng y G. Luo, "Progressive sampling-based Bayesian optimization for efficient and automatic machine learning model selection", *Health Inf Sci Syst*, vol. 5, n.º 2, 2017.

[24] A. Klein, S. Falkner, S. Bartels, P. Hennig y F. Hutter, *Fast Bayesian Optimization of Machine Learning Hyperparameters on Large Datasets*, 2017. arXiv: 1605.07079 [cs.LG].

[25] D. Choi, H. Cho y W. Rhee, "On the Difficulty of DNN Hyperparameter Optimization Using Learning Curve Prediction", en *TENCON 2018 - 2018 IEEE Region 10 Conference*, 2018, págs. 0651-0656. DOI: 10.1109/TENCON.2018.8650070.

[26] J. L. de la Fuente O'Connor, *Ingeniería de los Algoritmos y Métodos Numéricos*. Editorial Círculo Rojo., 2017, ISBN: 9788491409847. dirección: http://www.jldelafuenteoconnor.es/Libro2017_NV_10-8_SP.pdf.

[27] T. Amemiya, "Chapter 6 Non-linear regression models", en, ép. Handbook of Econometrics, vol. 1, Elsevier, 1983, págs. 333-389. DOI: https://doi.org/10.1016/S1573-4412(83)01010-7.

[28] T. Strutz, *Data Fitting and Uncertainty (A practical introduction to weighted least squares and beyond)*. oct. de 2010, ISBN: 3834810223.

[29] C. T. Kelley, *Iterative Methods for Optimization*. Society for Industrial y Applied Mathematics, 1999. DOI: 10.1137/1.9781611970920.

[30] P. I. Frazier, *A Tutorial on Bayesian Optimization*, 2018. arXiv: 1807.02811 [stat.ML].

[31] E. Brochu, V. M. Cora y N. de Freitas, *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*, 2010. arXiv: 1012.2599 [cs.LG].

[32] J. González, "Introduction to Bayesian Optimization", 2017. dirección: http://gpss.cc/gpss17/slides/gpss_bayesopt2017.pdf.

[33] T. Hastie, R. Tibshirani y J. Friedman, "Support Vector Machines and Flexible Discriminants", en *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, págs. 417-458, ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_12. dirección: https://doi.org/10.1007/978-0-387-84858-7_12.

[34] ——, "Neural Networks", en *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. New York, NY: Springer New York, 2009, págs. 389-416, ISBN: 978-0-387-84858-7. DOI: 10.1007/978-0-387-84858-7_11. dirección: https://doi.org/10.1007/978-0-387-84858-7_11.

[35] A. Ns, "Time complexity analysis of support vector machines (SVM) in LibSVM", *International Journal of Computer Applications*, vol. 128, págs. 975-8887, oct. de 2015. DOI: 10.5120/ijca2015906480.

[36] A. García Pérez, "Análisis de Componentes Principales", en *Métodos avanzados de estadística aplicada. Técnicas avanzadas*. UNED, 2005, págs. 25-62, ISBN: 978-8-436-25144-9.

A. Código R para la predicción de la curva de aprendizaje

El presente código ha sido tomado de [5].

```
Inverse Power Model.R
############################################################
############################################################
#need to set the following parameters
#functions to calculate rmse and mae should be defined by the user
#offset = offset to start fitting, it's always set to zero
#i = number of points to include in training data
#W = weights
#N = Total number of points
#startParams = start parameters to nls function currently
#              set to (a=0,b=1,c=-0.5)

FitModel<- function(offset, X, Y, W, i, N,startParams)
{
    #Data considering only points between offset and i
     x<-X[offset:i];
     y<-Y[offset:i];
     w<-W[offset:i];
    gradientF<-deriv3(~(1-a)-(b*(x^c)), c("a","b","c"),
                      function(a,b,c,x) NULL);
    # fitting the model using nls
    m<-nls(y~gradientF(a,b,c,x), start = startParams, weights=w,
             control = list(maxiter=1000, warnOnly = TRUE),
             algorithm = "port", upper = list(a=10, b = 10, c = -0.1),
             lower = list(a = 0, b = 0, c=-10),
             data = data.frame(y=y, x=x))
    #predict Y for sample sizes not used to fit the curve
    #if all data was used to fit model, testing data = training data
    #else, testing data  = (total data - training data)
    if (i==N){
        testX<-X[(offset:i)];
        testY<-Y[(offset:i)];
        testW <- W[offset:i]; }
```

```
else{
    testX<-X[((i+1):N)];    #Get remaining X
    testY<-Y[((i+1):N)];
    testW<-W[((i+1):N)];
    }
#predictions on unseen data
prediction<-predict(m, list(x=testX));
#confidence intervals
se.fit <- sqrt(apply(attr(predict(m, list(x=testX)),"gradient"),
               1,function(x) sum(vcov(m)*outer(x,x))));
prediction.ci <- prediction + outer(se.fit,qnorm(c(.5,.025,.975)));

predictY<-prediction.ci[,1];
predictY.lw<-prediction.ci[,2];
predictY.up<-prediction.ci[,3];

#Calculate residuals
if (i==N){
    res<-rep(0,length(X))
    }
else{
    res<-rep(0,length(X-i))
    }
res<-(predictY-testY);

#Calculate Root Mean square error

rmseValue<-rmse(testY, predictY);
#Calculate Absolute error

maeValue<-abse(testY, predictY);

}

#End function
```

B. Código R para la predicción de la curva de aprendizaje con SVM

```
scoring_function <- function(D,G,coef,cost,tol,eps,shri,
                             BBDD,P, k1, k2, k3, S,l, V)
{
  shri=ifelse(shri==1,TRUE,FALSE)
  N<-dim(training_set)[1]
  n<-(N*(P))-((N*(P))%%10)
  Nube_X<-seq(k1,k2,k3)
  Nube_Y<-c()
  Nube_n=length(Nube_X)
  for(x in Nube_X[1:Nube_n]){
    Accuracy<-c()
    for(i in seq(1,S)){
      set.seed(sample(1:1000000,1))
      split <- sample.split(BBDD[,V], SplitRatio = n/N)
      BBDD_val <- subset(BBDD, split == FALSE)
      BBDD_ent <- subset(BBDD, split == TRUE)
      if(x==n){
        BBDD_ent<-BBDD_ent
      }else{
        set.seed(sample(1:1000000,1))
        split <- sample.split(BBDD_ent[,V], SplitRatio = x/n)
        BBDD_ent <- subset(BBDD_ent, split == TRUE)
      }
      Model <- svm(formula = #NOMBRE DE LA VARIABLE V ~ .,
                   data = BBDD_ent,
                   type = "C-classification",
                   kernel = "polynomial",
                   degree=D,
                   gamma=G,
                   coef0=coef,
                   cost=cost,
                   tolerance=tol,
                   epsilon=eps,
                   shrinking=shri)
```

```
      y_result <- predict(Model, BBDD_val[,-V])
      Acc=Metrics::accuracy(actual=BBDD_val[,V],predicted=y_result)
      Accuracy<-c(Accuracy,Acc)
    }
    Nube_Y<-c(Nube_Y,mean(Accuracy))
  }
  Pesos<-c()
  for(i in 1:Nube_n){
    Pesos<-c(Pesos,i/Nube_n)
  }
  N=Nube_n
  startParams=c(a=0.1,b=1,c=-0.5)
  gradientF<-deriv3(~((1-a)-(b*(x^c))), c("a","b","c"), function(a,b,c,x) NULL)
  m<-nls(y~gradientF(a,b,c,x), start = startParams, weights=Pesos,
         control = list(maxiter=1000, warnOnly = TRUE),
         algorithm = "port", upper = list(a=10, b = 10, c = -0.1),
         lower = list(a = 0.001, b = 0.001, c=-10),
         data = data.frame(y=Nube_Y, x=Nube_X))
  A=summary(m)$coefficients
  Modelo<-function(x){((1-A[1])-(A[2]*(x^A[3])))}
  return(list(Score =Modelo(l)))
}
```

C. Código R para la predicción de la curva de aprendizaje con RN

```
scoring_function2 <- function(L1,L2,L3,LR,RHO,D,
                              BBDD,P,k1,k2,k3,S,l,V)
{
  N<-dim(BBDD)[1]
  n<-(N*(P))-((N*(P))%%10)
  Nube_X<-seq(k1,k2,k3)
  Nube_Y<-c()
  Nube_n=length(Nube_X)
  for(x in Nube_X[1:Nube_n]){
    Accuracy<-c()
    for(i in seq(1,S)){
      set.seed(sample(1:1000000,1))
      split <- sample.split(BBDD[,V], SplitRatio = n/N)
      BBDD_val <- subset(BBDD, split == FALSE)
      BBDD_ent <- subset(BBDD, split == TRUE)
      if(x==n){
        BBDD_ent<-BBDD_ent
      }else{
        set.seed(sample(1:1000000,1))
        split <- sample.split(BBDD_ent[,V], SplitRatio = x/n)
        BBDD_ent <- subset(BBDD_ent, split == TRUE)
      }
      train_x=as.matrix(BBDD_ent[,-V])
      train_y=as.matrix(BBDD_ent[,V])
      test_x=as.matrix(BBDD_val[,-V])
      test_y=as.matrix(BBDD_val[,V])
      model_2 = keras_model_sequential()
      NV= #NÚMERO VARIABLES ENTRADA
      model_2 %>%
        layer_dense(units = L1, activation = 'relu', input_shape =NV)%>%
        layer_dense(units = L2, activation = 'relu')%>%
        layer_dense(units = L3, activation = 'relu')%>%
        layer_dense(units = 1, activation = 'sigmoid')
      model_2 %>%
```

```
      compile (
        optimizer = optimizer_rmsprop(
          lr = LR,
          rho = RHO,
          epsilon = NULL,
          decay = D,
          clipnorm = NULL,
          clipvalue = NULL),
        loss = 'mean_squared_error',
        metrics = c('accuracy')
      )
    model_2 %>%
      fit (
        x = train_x,
        y = train_y,
        epochs = 25,
        batch_size = 10,
        verbose=0
      )
    Acc = model_2 %>%
      evaluate(
        test_x,
        test_y,
        verbose=0)

    Accuracy<-c(Accuracy,Acc[2])
  }
  Nube_Y<-c(Nube_Y,mean(Accuracy))
}
Pesos<-c()
for(i in 1:Nube_n){
  Pesos<-c(Pesos,i/Nube_n)
}
N=Nube_n
startParams=c(a=0.1,b=1,c=-0.5)
gradientF<-deriv3(~((1-a)-(b*(x^c))), c("a","b","c"), function(a,b,c,x) NULL)
m<-nls(y~gradientF(a,b,c,x), start = startParams, weights=Pesos,
       control = list(maxiter=1000, warnOnly = TRUE),
```

```
        algorithm = "port", upper = list(a=10, b = 10, c = -0.1),
        lower = list(a = 0.001, b = 0.001, c=-10),
        data = data.frame(y=Nube_Y, x=Nube_X))

  A=summary(m)$coefficients

  Modelo<-function(x){((1-A[1])-(A[2]*(x^A[3])))}
  return(list(Score =Modelo(l)))
}
```

D. Código R para la optimización bayesiana de SVM con la predición de las curvas de aprendizaje

```
limites <- list(
  # Emplear sufijo L si son valores enteros
  D  = c(1L, 5L),
  G   = c(2^-15,2^3),
  coef = c(-1,1),
  cost= c(1L,2L),
  tol=c(10^-5,10^-1),
  eps=c(0.001,0.5),
  shri=c(0L,1L)
)

cl <- makeCluster(8)
registerDoParallel(cl)
clusterExport(cl,c('BBDD'))
clusterEvalQ(cl,expr= {
  c(library(e1071),
  library(caret))
})


optObj <- bayesOpt(
    FUN = scoring_function # Apéndice B
    , bounds = limites
    , initPoints = 8
    , iters.n = 10
    , iters.k = 2
    , parallel = TRUE
    , acq = "ei"
)

stopCluster(cl)
registerDoSEQ()
```

E. Código R para la optimización bayesiana de RN con la predición de las curvas de aprendizaje

```
  limites <- list(
  # Emplear sufijo L si son valores enteros
  L1  = c(1L, 250L),
  L2   = c(1L, 250L),
  L3 = c(1L, 250L),
  LR= c(0,1),
  RHO=c(0,1),
  D=c(0,1)
)

optObj <- bayesOpt(
      FUN = scoring_function2 #Apéndice C
      , bounds = limites
      , initPoints = 10
      , iters.n = 4
      , iters.k = 2
      , parallel = FALSE
    )
```