USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA
FACULTADE DE MATEMÁTICAS

Traballo Fin de Grao

APRENDIZAJE ESTADÍSTICO PARA LA SELECCIÓN DE ALGORITMOS EN PROBLEMAS DE OPTIMIZACIÓN

Antonio Fariña Elorza

2023/2024

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

GRADO DE MATEMÁTICAS

Traballo Fin de Grado

APRENDIZAJE ESTADÍSTICO PARA LA SELECCIÓN DE ALGORITMOS EN PROBLEMAS DE OPTIMIZACIÓN

Antonio Fariña Elorza

Julio, 2024

UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Trabajo propuesto

Área de Conocimiento: Estadística e Investigación Operativa
Título: Aprendizaje estadístico para la selección de algoritmos en problemas de optimización
Breve descripción del contenido
El objetivo de este trabajo es estudiar técnicas de selección de algoritmos en el contexto de la resolución eficiente de problemas de optimización. Este trabajo se centrará en la descripción del problema de aprendizaje estadístico subyacente así como de distintas técnicas de regresión que se pueden aplicar para abordarlo. La segunda parte del trabajo consistirá en realizar algún estudio que permita comparar la calidad de las soluciones obtenidas con las distintas técnicas.
Recomendaciones
Otras observaciones

Índice

Resumen

En este trabajo se emplearán técnicas de aprendizaje estadístico para predecir el optimizador global que mejor funciona dado un problema de programación matemática no lineal. Antes de nada, se explicará un método de resolución de problemas de programación lineal entera y su adaptación al caso no lineal, donde surgirán nuevas dificultades. Posteriormente, se presentará el problema de aprendizaje estadístico y dos técnicas que permiten ajustar un modelo y crear predicciones: la regresión lineal y las redes neuronales de una sola capa oculta. Estas técnicas permitirán realizar el aprendizaje sobre un conjunto de problemas y obtener los resultados, viendo el desempeño de los distintos optimizadores.

Abstract

In this work, statistical learning techniques will be used to predict the best performing global optimiser for a non-linear mathematical programming problem. First of all, a method for solving integer linear programming problems and its adaptation to the nonlinear case, where new difficulties will arise, will be explained. Subsequently, the statistical learning problem and two techniques that allow to fit a model and create predictions will be presented: linear regression and single hidden layer neural networks. These techniques will allow learning to be performed on a set of problems and the results to be obtained, looking at the performance of the different optimisers.

Introducción

Hoy en día muchos problemas de grandes áreas de conocimiento como la economía, biología o ingeniería se pueden modelizar como problemas de programación matemática no lineal. En este campo se han llevado a cabo numerosos estudios y desarrollado varios optimizadores con el objetivo de resolver de manera eficiente este tipo de problemas, donde la complejidad de resolución crece de forma evidente al emplear funciones no lineales.

Muchos investigadores han centrado sus esfuerzos en los últimos años en desarrollar herramientas para permitir una resolución más rápida, mejorando los algoritmos existentes e incluso centrándose en un tipo de problema en concreto. Tras la realización de prácticas extracurriculares en el CITMAga la importancia y el interés en esta materia se ha vuelto evidente. Durante las prácticas se colaboró en el desarrollo una librería en R con el objetivo de aprender la mejor configuración del optimizador RAPOSa, centrado en la resolución de problemas de optimización polinómica. Esta librería permitía analizar un conjunto de problemas con sus características y predecir la estrategia que mejor resolvería un nuevo problema, mostrando de manera gráfica el rendimiento de cada una de ellas.

Este trabajo toma un nuevo enfoque en el aprendizaje previo, en el que ahora nos centraremos en problemas de programación matemática no lineal entera mixta (MINLP), donde RAPOSa podría no funcionar. Para introducir el tema se hará un breve recordatorio de la aproximación empleada para resolver los problemas de programación lineal entera mixta vista durante el Grado, el método de Ramificación y Acotación. El Capítulo 1 de este trabajo se centrará en este resumen, con el objetivo de establecer un punto de partida a la optimización no lineal. Será en el Capítulo 2 donde se verá la adaptación del método anterior al caso no lineal, de la mano del optimizador COUENNE. Esto se expondrá mediante el método de Ramificación y Acotación Espacial.

Los últimos capítulos de este trabajo estarán orientados al aprendizaje estadístico. En el Capítulo 3 se presentará de forma general esta rama de la estadística y se explicarán dos técnicas de aprendizaje que se usarán en el Capítulo 4: la regresión lineal y las redes neuronales de una capa oculta. Finalmente, el Capítulo 4 incluirá el estudio computacional y sus resultados numéricos asociados. El objetivo de este estudio será ver que tomar el optimizador que nos proporciona

el aprendizaje es mejor que tomar uno de manera general; el que mejor rendimiento promedio consigue.

Capítulo 1

Recordatorio de Programación Lineal y Entera

En este primer capítulo haremos un recordatorio de conceptos estudiados en la materia *Programación Lineal y Entera* del Grado en Matemáticas. Comenzaremos el capítulo con la definición de una serie de conceptos y notaciones que serán de utilidad, continuando con la explicación del método de resolución de problemas de programación lineal entera mixta, el **algoritmo de Ramificación y Acotación**. Todos estos conceptos se pueden consultar con más detalle en [5], pero con lo aquí introducido será suficiente de cara a los capítulos siguientes.

1.1. Preliminares

Comenzaremos entonces esta sección con un breve recordatorio de algunos conceptos vistos en el grado y que serán de utilidad más adelante.

Definición 1.1. (Conjunto convexo). Sea $A \subset \mathbb{R}^n$ un conjunto. Decimos que A es *convexo* si $\forall \boldsymbol{x}, \boldsymbol{y} \in A$ se tiene que

$$\{\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y} : \lambda \in [0,1]\} \subset A.$$

Definición 1.2. (Función convexa). Se dice que $f : \mathbb{R}^n \to \mathbb{R}$ es una *función convexa* si dados $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ y $\lambda \in [0,1]$ se tiene que

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{y}) \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{y}).$$

Decimos que f es estrictamente convexa si se da la desigualdad estricta en el caso anterior, con $\boldsymbol{x} \neq \boldsymbol{y}$ y $\lambda \in (0,1)$.

Definición 1.3. (Función afín y función lineal). Sea $f : \mathbb{R}^n \to \mathbb{R}$. Decimos que f es una *función afín* si es de la forma

$$f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} + b = a_1x_1 + a_2x_2 + \cdots + a_nx_n + b,$$

con $\boldsymbol{a} \in \mathbb{R}^n$ y $b \in \mathbb{R}$.
En particular, si $b = 0$ diremos que f es una *función lineal*.

A continuación veremos que toda función afín (y por lo tanto lineal) es convexa, ya que si f es afín, $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ y $\lambda \in [0, 1]$ se tiene

$$\begin{aligned} f(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) = \boldsymbol{a}^T(\lambda\boldsymbol{x} + (1-\lambda)\boldsymbol{y}) + b = \lambda(\boldsymbol{a}^T\boldsymbol{x}) + (1-\lambda)(\boldsymbol{a}^T\boldsymbol{y}) + \lambda b + (1-\lambda)b = \\ = \lambda(\boldsymbol{a}^T\boldsymbol{x} + b) + (1-\lambda)(\boldsymbol{a}^T\boldsymbol{y} + b) = \lambda f(\boldsymbol{x}) + (1-\lambda)f(\boldsymbol{y}). \end{aligned}$$

Continuaremos formulando los conceptos de problema de optimización y programación matemática, claves en el desarrollo de este capítulo.

Definición 1.4. (Problema de optimización) Un *problema de optimización* viene dado por un par (F, f) donde F es el conjunto de los puntos factibles (o región factible) y f es la función objetivo o de coste. El problema consiste en encontrar un punto factible $\boldsymbol{x} \in F$ tal que $f(\boldsymbol{x}) \leq f(\boldsymbol{y})\ \forall \boldsymbol{y} \in F$. Un punto en estas condiciones es un *óptimo global* del problema.

Definición 1.5. (Problema de programación matemática) Un problema de programación matemática es un problema de optimización en el que el conjunto factible F viene dado por unas funciones de restricción $g_j,\ j \in M = \{1, \ldots, m\}$ y $h_k,\ k \in L = \{1, \ldots, l\}$ y que encaja en la siguiente formulación

$$\begin{aligned} \text{minimizar} \quad & f(\boldsymbol{x}) \\ \text{sujeto a} \quad & g_j(\boldsymbol{x}) \leq 0 \qquad j \in M \\ & h_k(\boldsymbol{x}) = 0 \qquad k \in L. \end{aligned}$$

En este trabajo nos centraremos en los problemas de programación matemática con región factible acotada.

A la hora de resolver los problemas anteriores podríamos pensar en obtener la solución basándonos en condiciones locales, pero esto en la mayoría de los casos provocaría la obtención de óptimos locales, que pueden estar muy distantes del óptimo global del problema. Encontrar un óptimo global es en general complejo. Sin embargo, si se dan unas condiciones específicas estos conceptos serán equivalentes y podremos encontrar el óptimo global de un problema basándonos solo en condiciones locales. Se trata de los *problemas de programación convexa*.

Definición 1.6. (Problema de programación convexa) Un *problema de programación convexa* es un problema de programación matemática en el que las funciones f y g_j son convexas y las funciones h_k son afines (o lineales).

Para los problemas anteriores el conjunto factible F es un conjunto convexo. Si además las funciones f y g_j son lineales tendremos un problema de programación lineal, para los que existen numerosos métodos de resolución eficientes, como el método símplex, capaces de resolver en tiempo polinomial cualquier problema de manera exacta. Dado que todas las funciones que intervienen en el problema son lineales podemos formularlo como

$$\begin{aligned} \text{minimizar} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{sujeto a} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \end{aligned} \tag{LP}$$

donde $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ y $\boldsymbol{b} \in \mathbb{R}^m$

En muchos casos necesitaremos obtener soluciones a problemas en los que alguna variable deba tomar valores enteros. Estos problemas son los problemas de programación lineal entera mixta (*MILP* en inglés).

Definición 1.7. (Mixed Integer Linear Programming MILP) Un *problema de programación lineal entera mixta* es un problema de programación lineal en el que algunas variables deben tomar valores enteros, es decir

$$\begin{aligned} \text{minimizar} \quad & \boldsymbol{c}^T\boldsymbol{x} \\ \text{sujeto a} \quad & \boldsymbol{A}\boldsymbol{x} \leq \boldsymbol{b} \\ & \boldsymbol{x} \geq \boldsymbol{0} \\ & x_i \in \mathbb{Z} \qquad \text{para } i \in N_I \subseteq \{1,\dots,n\}. \end{aligned} \tag{MILP}$$

con $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m\times n}$ y $\boldsymbol{b} \in \mathbb{R}^m$

En este tipo de problemas la región factible ya no es un conjunto convexo y la complejidad de resolución aumenta considerablemente. Para atacar este tipo de problemas se han desarrollado numerosos algoritmos, como el **algoritmo de Ramificación y Acotación** que explicaremos en la siguiente sección.

1.2. Resolución de MILP mediante Ramificación y Acotación

Como comentábamos anteriormente, este tipo de problemas tienen una resolución más difícil que los problemas de programación lineal, en los que la región factible era convexa. Ahora, cuando

se requiere que alguna variable sea entera la convexidad se pierde y los métodos enumerativos resultan tremendamente ineficientes.

La idea del algoritmo de Ramificación y Acotación es resolver sucesivamente numerosas versiones relajadas del problema original, formulado como (MILP), en las que no se tiene en cuenta la restricción de integralidad de las variables enteras. En estas versiones relajadas el valor de la función objetivo en el óptimo proporcionará una cota inferior para el óptimo del problema MILP y cualquier punto factible una cota superior. Recordemos que nos encontramos en el caso de que la región factible es acotada, aunque esta restricción no es limitante, pues en la mayoría de problemas reales las variables a tener en cuenta no pueden tomar cualquier valor.

Antes de mostrar el esquema del algoritmo definiremos la *medida de infactibilidad para variables enteras* $U_i(\boldsymbol{x}) := \min(x_i - \lfloor x_i \rfloor, \lceil x_i \rceil - x_i)$, donde $\lfloor x_i \rfloor$ denota al mayor entero menor o igual que x_i y $\lceil x_i \rceil$ denota al menor entero mayor o igual que x_i. Incluimos el esquema en la Tabla 1.1.

Entrada: problema de programación lineal entera mixta MILP
Definir LP_1 como el problema relajado de $MILP$ y su potencial $LB_1 = -\infty$ Inicializar la cota inferior $LB = -\infty$ y superior $UB = +\infty$ Inicializar la mejor solución hasta el momento $\boldsymbol{x}^{best} = \emptyset$, $\mathcal{L} = \{1\}$ y $tot = 1$ **mientras** $\mathcal{L} \neq \emptyset$ y $LB \neq UB$ elegir el menor $k \in \mathcal{L}$ tal que $LB_k = \min_{s \in \mathcal{L}} LB_s$ y actualizar $\mathcal{L} = \mathcal{L} \backslash \{k\}$ resolver el problema lineal LP_k **si** LP_k es factible, **entonces** obtener $\bar{\boldsymbol{x}}^k$ solución de LP_k y $\bar{z}^k = f(\bar{\boldsymbol{x}}^k)$ **si** $\bar{z}^k < UB$ y $\bar{x}_i^k \in \mathbb{Z}$ con $i \in N_I$ **entonces** actualizar $UB = \bar{z}^k$ y $\boldsymbol{x}^{best} = \bar{\boldsymbol{x}}^k$ eliminar los $k \in \mathcal{L}$ tales que $LB_k \geq UB$ (ACOTACIÓN) **si** $\bar{z}^k < UB$ y $\exists i \in N_I$ tal que $\bar{x}_i^k \notin \mathbb{Z}$ **entonces** tomar el menor $i \in N_I$ con mayor valor para $U_i(\bar{\boldsymbol{x}}^k) = \min(\bar{x}_i^k - \lfloor \bar{x}_i^k \rfloor, \lceil \bar{x}_i^k \rceil - \bar{x}_i^k)$ ramificar creando los subproblemas (RAMIFICACIÓN) LP_{tot+1} resultado de añadir a LP_{tot} la restricción $x_i \leq \lfloor \bar{x}_i^k \rfloor$ con potencial $LB_{tot+1} = \bar{z}^k$ LP_{tot+2} resultado de añadir a LP_{tot} la restricción $x_i \geq \lceil \bar{x}_i^k \rceil$ con potencial $LB_{tot+2} = \bar{z}^k$ actualizar $\mathcal{L} = \mathcal{L} \cup \{tot+1, tot+2\}$ y $tot = tot + 2$ **si** $\bar{z}^k \geq UB$ se descarta el problema (ACOTACIÓN) **si** $\mathcal{L} \neq \emptyset$ actualizamos $LB = \min_{s \in \mathcal{L}} LB_s$
Salida: solución óptima $\boldsymbol{x}^{best}$ y UB su valor objetivo

Tabla 1.1: Esquema del algoritmo de Ramificación y Acotación.

El algoritmo de Ramificación y Acotación comienza con la inicialización de una serie de parámetros y la creación de LP_1, relajación del problema original (MILP). Posteriormente se toma el problema con menor potencial de los pendientes por resolver y se resuelve. En el caso de que sea factible se obtiene la solución $\bar{\boldsymbol{x}}^k$ y el valor del objetivo $\bar{z}^k$. Ahora se tienen 3 escenarios:

- $\bar{z}^k < UB$ y $\bar{\boldsymbol{x}}^k$ cumple las restricciones de integralidad: tenemos un problema factible con solución con menor potencial que el actual (dado por UB) por lo que se actualiza UB y $\boldsymbol{x}^{best}$ y se descartan por acotación aquellos problemas que no puedan alcanzar un valor mejor (aquellos tales que $LB_k \geq UB$).
- $\bar{z}^k < UB$ y algún $\bar{x}_i^k$ no es entero cuando debería: ramificamos, eligiendo como variable de ramificación de entre las que no cumplen ser enteras aquella que más lejos esté de serlo (mayor U_i) . Creamos dos subproblemas dividiendo la región factible de LP_{tot} en dos y actualizamos $\mathcal{L}$.
- $\bar{z}^k \geq UB$: este problema proporciona un valor del objetivo mayor o igual que la cota superior (peor valor que se puede conseguir) por lo que se descarta.

Finalmente se actualiza la cota inferior con el menor potencial de entre los problemas pendientes de resolver. Este procedimiento se repite hasta que no queden problemas por resolver o hasta que las cotas coincidan, obteniendo como solución $\boldsymbol{x}^{best}$ y como valor objetivo UB.

En el algoritmo anterior se han tenido en cuenta una serie de decisiones en cuanto a la resolución del problema. En primer lugar, se elige el problema a resolver $k \in \mathcal{L}$ con menor potencial de entre los pendientes. La elección del problema podría ser arbitraria, incluso siguiendo una estrategia en anchura o profundidad, pero habitualmente escoger el que mejor potencial tiene resulta en una convergencia más rápida del algoritmo. Por otra parte, se elige como variable de ramificación x_i aquella más lejos de ser entera, la que maximiza $U_i(\bar{x}^k) := \min(\bar{x}_i^k - \lfloor \bar{x}_i^k \rfloor, \lceil \bar{x}_i^k \rceil - \bar{x}_i^k)$.

Veremos una adaptación de este algoritmo en el siguiente capítulo, para problemas en el que las restricciones son no lineales.

Capítulo 2

Resolución de problemas de programación no lineal entera mixta

Una vez visto en el capítulo anterior un repaso de ciertas cuestiones vistas en el Grado entraremos ahora en materia nueva, viendo como se adapta el **algoritmo de Ramificación y Acotación** a la resolución de problemas de programación no lineal entera mixta (*MINLP* en inglés). En este capítulo se explicará el **algoritmo de Ramificación y Acotación Espacial** utilizado por varios optimizadores, pero centrándonos en el caso de COUENNE. Se explicarán los aspectos más importantes, pudiendo entrar más en detalle en [4].

2.1. Idea de las técnicas de optimización global

Si recordamos lo que pasaba en el caso lineal entero, el algoritmo de Ramificación y Acotación resolvía sucesivamente relajaciones lineales del problema, sin tener en cuenta la restricción de integralidad. Lo que se buscaba con estas relajaciones era convertir la región factible en una región convexa, para la que el algoritmo del símplex podía obtener una solución de manera eficiente, siendo esta además, global. Era precisamente la convexidad de la región factible lo que nos garantizaba el carácter global de la solución.

En este capítulo buscaremos resolver problemas con la formulación

$$\begin{array}{rll} \min & f(\boldsymbol{x}) & \\ \text{sujeto a} & g_j(\boldsymbol{x}) \leq 0 & \forall j \in M \\ & x_i^l \leq x_i \leq x_i^u & \forall i \in N_0 \\ & x_i \in \mathbb{Z} & \forall i \in N_0^I \subseteq N_0, \end{array} \tag{NLP}$$

donde $n = |N_0|$ es el número de variables del problema y $\boldsymbol{x} = (x_i)_{i \in N_0}$ es el vector n-dimensional de variables. En la formulación (NLP) se han omitido las restricciones de igualdad, pues nos centraremos en problemas sin este tipo de restricciones. Sin embargo, los argumentos que se expondrán serán igualmente válidos en el caso de incluirlas.

Si el problema (NLP) es no lineal pero convexo tenemos garantizada la optimalidad global, basándonos solo en condiciones locales. Esta propiedad nos permite resolver de manera eficiente el problema con multitud de algoritmos. Esta idea es la que explotan los algoritmos de Ramificación y Acotación Espacial para resolver problemas generales (no convexos, en muchos casos).

De igual manera que en el caso lineal el algoritmo de Ramificación y Acotación creaba relajaciones LP lineales, la adaptación a los problemas NLP pasa por resolver una sucesión de subproblemas convexos del problema de partida. En este ámbito se han realizado numerosos estudios y se han desarrollado varios optimizadores globales como BARON [7] y LINDOGLOBAL [9]. Además los optimizadores para problemas convexos se pueden usar para aquellos no convexos como heurísticas, ya que podrían producir una solución factible para el problema no convexo. COUENNE (**C**onvex **O**ver and **U**nder **EN**velopes for **N**onlinear **E**stimation) [3], implementa la Ramificación y Acotación Espacial centrándose en el caso en el que las funciones son no convexas.

La idea de este algoritmo consiste en resolver, a lo largo del árbol de ramificación y acotación, subproblemas convexos, para los que hay garantía de optimalidad global. En particular, COUENNE resuelve problemas lineales en todos los nodos del árbol. Para poder llevar a cabo esto será necesario, igual que en el caso lineal, poder crear las relajaciones lineales de las restricciones de partida, para lo que inicialmente se reescribirá el problema de una manera más sencilla, mediante lo que definiremos como *funciones factorizables*. Una vez reescrito el problema será mucho más fácil atacar cada una de las restricciones (posiblemente no lineales) que delimitan la región factible, reemplazándolas por restricciones lineales que definen regiones poliédricas conteniendo a todas las soluciones del problema original. De aquí viene el nombre de COUENNE, ya que las regiones que delimitan las distintas restricciones "se envuelven superiormente e inferiormente" mediante conjuntos convexos, normalmente poliedros.

A medida que el algoritmo avance se irá ramificando más y más, y estas relajaciones convexas aproximarán cada vez mejor la región factible en el nodo correspondiente del árbol de ramificación y acotación. Esto permitirá al algoritmo asegurar que, en el límite, se encontrará el óptimo global del problema.

En la siguiente sección veremos un esquema del algoritmo completo y nos centraremos en la generación de las relajaciones lineales, comentando brevemente también la ramificación.

2.2. Algoritmo de Ramificación y Acotación Espacial

Explicaremos a continuación el esquema del algoritmo. Consideremos entonces un MINLP de la forma (NLP) suponiendo las funciones f y g_j, $j \in M$ no necesariamente convexas y factorizables. Comenzaremos definiendo este último concepto, que tomará importancia en la reformulación del problema.

Definición 2.1. (Función factorizable) Una *función factorizable* $t(\boldsymbol{x})$ es una función que devuelve la coordenada i-ésima de $\boldsymbol{x} \in \mathbb{R}^n$, $t(\boldsymbol{x}) = x_i$, o aquella que se puede obtener usando alguna operación de las siguientes sobre otras funciones factorizables $t_j(\boldsymbol{x})$:

- una combinación lineal, $t(\boldsymbol{x}) = a_0 + \sum_{j=1}^{k} a_j t_j(\boldsymbol{x})$;
- un producto, $t(\boldsymbol{x}) = \prod_{j=1}^{k} t_j(\boldsymbol{x})$;
- un cociente, $t(\boldsymbol{x}) = t_1(\boldsymbol{x})/t_2(\boldsymbol{x})$;
- una potencia, $t(\boldsymbol{x}) = t_1(\boldsymbol{x})^{t_2(\boldsymbol{x})}$;
- composición de funciones de una variable, $t(\boldsymbol{x}) = t_1(t_2(\boldsymbol{x}))$ (seno, logaritmo, exponencial, valor absoluto...).

Denotaremos por Θ al conjunto de operaciones anteriores,

$$\Theta = \{\text{suma, producto, cociente, potencia, exponencial, log, seno, coseno, abs}\}.$$

Podemos ver el esquema del algoritmo en la Tabla 2.1. Se trata de una adaptación del empleado para problemas MILPs, con algunas diferencias, que se han incluido en azul:

- Se introduce la reducción de cotas, que tiene como objetivo reducir la región factible sin cambiar la solución del problema. Se trata de un paso opcional pero puede mejorar notablemente el tiempo de ejecución del algoritmo. Como resultado de este paso es necesario verificar la factibilidad del problema obtenido, pudiendo obtener un problema no factible.
- La generación de la relajación lineal del problema en este caso será más complicada que para MILPs y dependerá del carácter de las funciones que delimitan la región factible. Lo abordaremos en las subsecciones 2.2.1 y 2.2.2.
- Se repite la obtención de la solución un número de veces, mediante el refinamiento de la linealización. Esto tiene como objetivo mejorar la solución obtenida.

- Dado de que la relajación lineal crea una aproximación lineal de la región factible hay que comprobar si la solución obtenida es factible para el problema original, con sus restricciones posiblemente no lineales (condición de factibilidad). En el caso lineal entero simplemente se comprobaba que las componentes enteras de la solución lo fuesen.

- Ramificación, que ahora podrá ser necesaria sobre variables continuas y se dará una idea de la misma en la subsección 2.2.3.

Entrada: problema de programación no lineal entera mixta P_1
Definir el potencial de P_1, $LB_1 = -\infty$ Inicializar la cota inferior $LB = -\infty$ y superior $UB = +\infty$ Inicializar la mejor solución hasta el momento $\boldsymbol{x}^{best} = \emptyset$, $\mathcal{L} = \{1\}$ y $tot = 1$ **mientras** $\mathcal{L} \neq \emptyset$ y $LB \neq UB$ elegir el menor $k \in \mathcal{L}$ tal que $LB_k = \min_{s\in\mathcal{L}} LB_s$ y actualizar $\mathcal{L} = \mathcal{L}\backslash\{k\}$ aplicar reducción de cotas a P_k (opcional) **si** P_k es factible, **entonces** generar la relajación lineal LP_k de P_k (Subsec. 2.2.1 y 2.2.2) **hacer** obtener $\bar{\boldsymbol{x}}^k$ solución de LP_k y $\bar{z}^k$ el valor de la función objetivo en $\bar{\boldsymbol{x}}^k$ refinar la linealización LP_k **hasta que** $\bar{z}^k$ no mejore lo suficiente **si** $\bar{z}^k < UB$ y $\bar{\boldsymbol{x}}^k$ es factible para P_k **entonces** actualizar $UB = \bar{z}^k$ y $\boldsymbol{x}^{best} = \bar{\boldsymbol{x}}^k$ eliminar los $k \in \mathcal{L}$ tales que $LB_k \geq UB$ (ACOTACIÓN) **si** $\bar{z}^k < UB$ y $\bar{\boldsymbol{x}}^k$ no es factible para P_k **entonces** elegir una variable x_i y un punto de ramificación x_i^b crear subproblemas: P_{tot+1} añadiendo la restricción $x_i \leq x_i^b$ con potencial $LB_{tot+2} = \bar{z}^k$ P_{tot+2} añadiendo la restricción $x_i \geq x_i^b$ con potencial $LB_{tot+2} = \bar{z}^k$ actualizar $\mathcal{L} = \mathcal{L} \cup \{tot+1, tot+2\}$ y $tot = tot + 2$ **si** $\bar{z}^k \geq UB$ se descarta el problema (ACOTACIÓN) **si** $\mathcal{L} \neq \emptyset$ actualizamos $LB = \min_{s\in\mathcal{L}} LB_s$
Salida: solución óptima $\boldsymbol{x}^{best}$ y UB su valor objetivo

Tabla 2.1: Esquema del Algoritmo de Ramificación y Acotación Espacial.

Comenzaremos de manera natural por explicar el proceso de linealización del problema, que consta de dos etapas: reformulación y linealización.

2.2.1. Reformulación

Esta primera etapa solo se realiza una vez, sobre el problema original, y consiste en expresar las restricciones y función objetivo del problema de una manera más sencilla, a costa de aumentar el número de variables. En este paso es donde toman importancia las funciones factorizables, definidas al inicio de la sección.

Tomando una función $t(\boldsymbol{x})$ factorizable, su reformulación se realiza de la siguiente manera: si $t(\boldsymbol{x})$ se puede obtener mediante funciones factorizables $t_j(\boldsymbol{x})$ con $j = 1, \ldots, k$, se añade una nueva variable w_{n+j} y la restricción $w_{n+j} = t_j(\boldsymbol{x})$ con $j = 1, \ldots, k$. Este proceso se repite de manera recursiva a todas las $t_j(\boldsymbol{x})$ hasta que todas las variables auxiliares introducidas estén asociadas con operadores del conjunto Θ. De esta manera podremos definir cada una de ellas como $w_{n+j} = \vartheta_j(\boldsymbol{x}, w_{n+1}, w_{n+2}, \ldots, w_{n+j-1})$ con $\vartheta_j \in \Theta$. Esta reformulación convierte el problema original en un problema en un espacio de mayor dimensión, al añadir las variables auxiliares con índices en $Q = \{n+1, n+2, \ldots, n+q\}$.

En términos generales el problema resultante toma la siguiente expresión

$$
\begin{aligned}
\min \quad & w_{n+q} \\
\text{s. a.} \quad & w_i = \vartheta_i(\boldsymbol{x}, w_{n+1}, w_{n+2}, \ldots, w_{i-1}) && i \in Q \\
& w_i^l \leq w_i \leq w_i^u && i \in Q \\
& x_i^l \leq x_i \leq x_i^u && i \in N_0 \\
& x_i \in \mathbb{Z} && i \in N_0^I \subseteq N_0 \\
& w_i \in \mathbb{Z} && i \in Q_I \subseteq Q,
\end{aligned}
$$

donde $\vartheta_i \in \Theta$, $\forall i \in Q$ y la función objetivo se reemplaza por la última variable auxiliar generada, w_{n+q}.

Finalizado el proceso de reformulación, podremos ignorar el carácter auxiliar u original de las variables, teniendo un problema MINLP con más variables y restricciones que el original pero con una estructura más sencilla:

$$
\begin{aligned}
\min \quad & x_{n+q} \\
s.a. \quad & x_i = \vartheta_i(\boldsymbol{x}) \quad i \in Q \subseteq N \\
& x_i^l \leq x_i \leq x_i^u \quad i \in N \\
& x_i \in \mathbb{Z} \quad i \in N_I \subseteq N,
\end{aligned}
$$

con $\vartheta_i \in \Theta$, $\forall i \in Q$, N englobando tanto las variables originales como las auxiliares y N_I las variables de carácter entero.

A continuación se mostrará el proceso anterior para el problema de ejemplo

$$\begin{aligned} \min \quad & -5x_1 - 2x_2 \\ \text{sujeto a} \quad & x_1^3 - x_1^2 + x_2 \leq 4 \\ & x_1 - x_2^2 \leq 0 \\ & -2 \leq x_1 \leq 2 \\ & 0 \leq x_2 \leq 4 \\ & x_1 \in \mathbb{Z}, \end{aligned} \tag{EJ}$$

cuya región factible podemos ver en la Figura 2.1. En rojo se muestra la región factible con la restricción de integralidad $x_1 \in \mathbb{Z}$, aunque también se ha sombreado en gris la región sin tener en cuenta esta restricción.

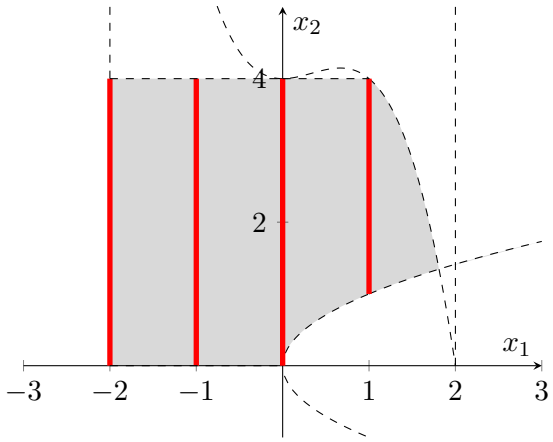


Figura 2.1: Región factible de (EJ).

En el ejemplo anterior tenemos dos variables x_1 y x_2, una función objetivo $f(x_1, x_2) = -5x_1 - 2x_2$ lineal y dos restricciones polinómicas. Además, las variables están acotadas y x_1 debe tomar valores enteros.

Comenzamos definiendo las variables $w_3 = x_1^3$ y $w_4 = x_1^2$ para las que podemos inferir cotas a partir de las cotas de x_1, lo que resulta en $w_3 \in [-8, 8] \cap \mathbb{Z}$ y $w_4 \in [0, 4] \cap \mathbb{Z}$. A partir de estas variables auxiliares podemos definir la variable $w_5 = x_2 + w_3 - w_4$, con $w_5 \in [-12, 12]$ y la restricción $w_5 \leq 4$ (estas dos restricciones se pueden combinar en $w_5 \in [-12, 4]$). Posteriormente, definimos la variable $w_6 = x_2^2$, con $w_6 \in [0, 16]$ y con ella la variable $w_7 = x_1 - w_6$, con $w_7 \leq 0$ y $w_7 \in [-18, 2]$ (que se combinan en $w_7 \in [-18, 0]$). Finalmente, definimos $w_8 = -5x_1 - 2x_2$ con $w_8 \in [-18, 10]$, que será la variable a minimizar.

Una vez reformulado el problema podemos ignorar el carácter original o auxiliar de las va-

riables y obtener la siguiente expresión equivalente

$$
\begin{aligned}
\min \quad & x_8 \\
\text{sujeto a} \quad & x_3 = x_1^3 \\
& x_4 = x_1^2 \\
& x_5 = x_2 + x_3 - x_4 \\
& x_6 = x_2^2 \\
& x_7 = x_1 - x_6 \\
& x_8 = -5x_1 - 2x_2 \\
-2 \leq\ & x_1 \leq 2 \\
0 \leq\ & x_2 \leq 4 \\
-8 \leq\ & x_3 \leq 8 \\
0 \leq\ & x_4 \leq 4 \\
-12 \leq\ & x_5 \leq 4 \\
0 \leq\ & x_6 \leq 16 \\
-18 \leq\ & x_7 \leq 0 \\
-18 \leq\ & x_8 \leq 10 \\
& x_1, x_3, x_4 \in \mathbb{Z}.
\end{aligned}
\tag{EJ'}
$$

En el Anexo I se incluye código para la resolución de los problemas anteriores mediante AMPL.

Tras este proceso de reformulación se realiza la linealización, que describimos en detalle en la siguiente subsección.

2.2.2. Linealización

Como se introdujo al inicio del capítulo, un aspecto central de los algoritmos de ramificación y acotación espacial es la definición de aproximaciones lineales o convexas del problema que se pretende resolver en cada nodo del árbol. Una vez que se dispone de una reformulación como la presentada en el apartado anterior, resulta relativamente sencillo acometer esta tarea restricción a restricción: cada una se reemplaza por un conjunto de restricciones lineales y/o convexas que envuelven la región factible delimitada por la región original. A continuación ilustraremos este procedimiento con una restricción sencilla del problema (EJ), partiendo de su versión reformulada, para luego explicar con rigor el método general utilizado.

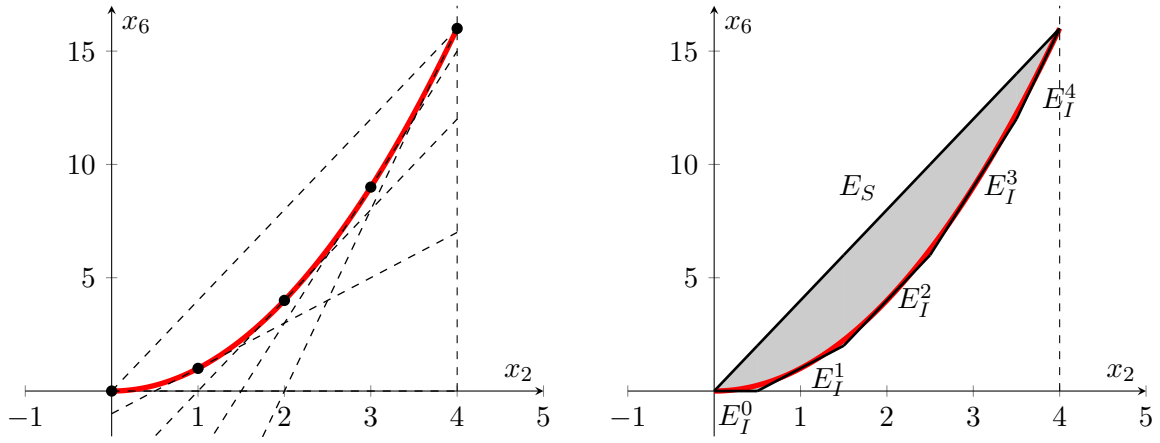
Siguiendo con la notación introducida, tomamos la restricción $x_6 = \vartheta_6(x_2) = x_2^2$ con $x_2 \in [0, 4]$

del problema (EJ'). Tenemos que envolver tanto inferiormente como superiormente la restricción por un conjunto convexo. Dado que tenemos $\vartheta_6 : \mathbb{R} \to \mathbb{R}$ podemos crear la envoltura mediante rectas que delimiten la gráfica de ϑ_6. Para la envoltura superior, E_S, podemos tomar la recta que pasa por los puntos $(0, \vartheta_6(0))$ y $(4, \vartheta_6(4))$, es decir $(0, 0)$ y $(4, 16)$. Esto se traduce en la siguiente desigualdad $x_6 \leq 4x_2$.

Por otra parte, para delimitar la región inferiormente podemos tomar rectas tangentes a la función en puntos fijados $(\tilde{x}_2, \vartheta_6(\tilde{x}_2))$ con $\tilde{x}_2 \in [0, 4]$. Estas vendrán dadas por la expresión $\vartheta_6(\tilde{x}_2) + \dot{\vartheta}_6(\tilde{x}_2)(x_2 - \tilde{x}_2)$. Tomando $\tilde{x}_2 \in \{0, 1, 2, 3, 4\}$ ya que $\dot{\vartheta}_6(x_2) = 2x_2$ tendremos las envolturas lineales inferiores, E_I^k con $k \in \{0, 1, 2, 3, 4\}$, siguientes

$$
\begin{aligned}
E_I^0 : &\quad x_6 \geq 0^2 + 2 \cdot 0(x_2 - 0) &\implies &\quad x_6 \geq 0 \\
E_I^1 : &\quad x_6 \geq 1^2 + 2 \cdot 1(x_2 - 1) &\implies &\quad x_6 \geq 2x_2 - 1 \\
E_I^2 : &\quad x_6 \geq 2^2 + 2 \cdot 2(x_2 - 2) &\implies &\quad x_6 \geq 4x_2 - 4 \\
E_I^3 : &\quad x_6 \geq 3^2 + 2 \cdot 3(x_2 - 3) &\implies &\quad x_6 \geq 6x_2 - 9 \\
E_I^4 : &\quad x_6 \geq 4^2 + 2 \cdot 4(x_2 - 4) &\implies &\quad x_6 \geq 8x_2 - 8.
\end{aligned}
$$

Juntando las restricciones definidas arriba tendremos la envoltura lineal y por lo tanto convexa deseada, cuya gráfica podemos ver en la Figura 2.2. En 2.2a podemos ver la función representada en rojo y las rectas que conforman la linealización con trazo discontinuo, tangentes en los puntos representados sobre la curva. En 2.2b tenemos representadas las restricciones lineales que delimitan la envoltura con trazo continuo negro, la restricción original contenida en la región en rojo, y su contenido sombreado en gris.



(a) Rectas de linealización sobre x_2^2.

(b) Resultado de la linealización.

Figura 2.2: Linealización para $x_6 = x_2^2$.

El procedimiento detrás de COUENNE consiste el repetir lo anterior para todas las res-

tricciones del problema. De manera general, partiendo de una variable $x_j = \vartheta_j(\boldsymbol{x})$, creada en el proceso de reformulación y las cotas $B = [\boldsymbol{x}^l, \boldsymbol{x}^u]$ sobre $\boldsymbol{x}$ se buscarán desigualdades de la forma $\boldsymbol{a}\boldsymbol{x} \geq b$ para cada $x_j = \vartheta_j(\boldsymbol{x})$ que se satisfagan para todos los elementos del conjunto $\{\boldsymbol{x} \in B : x_j = \vartheta_j(\boldsymbol{x})\}$. Esto es, buscaremos desigualdades lineales que envuelvan inferiormente la gráfica de $\vartheta_j(\boldsymbol{x})$. Así, para x_j tendremos un sistema de desigualdades lineales dado por $A^j\boldsymbol{x} \geq b^j$ que conformarán la envoltura inferior de la restricción. En el caso de tener una restricción procedente de la reformulación $x_j = \vartheta_j(\boldsymbol{x})$, $\boldsymbol{x} \in B$, con ϑ_j continuamente diferenciable y convexa el sistema de desigualdades lineales vendrá dado por hiperplanos de la forma $\vartheta_j(\tilde{\boldsymbol{x}}^p) + \nabla\vartheta_j(\tilde{\boldsymbol{x}}^p)^T(\boldsymbol{x} - \tilde{\boldsymbol{x}}^p)$, con $\tilde{\boldsymbol{x}}^p \in B$ fijados. El número de puntos a tomar es un parámetro que acepta COUENNE.

El proceso de linealización se repite para cada variable $x_j = \vartheta_j(\boldsymbol{x}),\ j \in Q$, obteniendo el problema LP_1 definido como $\min\{x_{n+q} : A\boldsymbol{x} \geq \boldsymbol{b}\}$ que será la relajación lineal de P_1. Además la solución óptima $\bar{\boldsymbol{x}}$ de LP_1 proporcionará una cota inferior $\bar{x}_{n+q}$ para el valor óptimo de P_1, en el caso de que sea factible. En caso contrario habrá que refinar la linealización añadiendo nuevas desigualdades lineales que separen la solución obtenida de la región factible. Si esto no es posible será necesario ramificar.

2.2.3. Ramificación

Comentaremos brevemente cómo se puede realizar la ramificación observando el paralelismo con el caso lineal. Las decisiones a tomar serán la elección de la variable de ramificación x_i y el punto usado para ello x_i^b.

Si tenemos una solución $\bar{\boldsymbol{x}}^k$ y la variable $x_i \in \mathbb{Z}$ pero $\bar{x}_i^k \notin \mathbb{Z}$ podemos tomar x_i como variable a ramificar, igual que en el caso lineal. No obstante, puede tener que ramificarse una variable continua, por lo que tendremos la medida de infactibilidad para variables continuas $U_i(\bar{\boldsymbol{x}}^k) := \frac{|\bar{x}_i^k - \vartheta_i(\bar{\boldsymbol{x}}^k)|}{1+||\nabla\vartheta_i(\bar{\boldsymbol{x}}^k)||_2}$. [1] Con esta medida un problema MINLP será factible si todas las variables enteras toman valores enteros y $U_i(\bar{\boldsymbol{x}}^k) = 0$ para todas las variables.

COUENNE utiliza las técnicas estándar para MILPs cuando hay que seleccionar una variable entera para ramificar. Para el caso de tener que ramificar en una variable continua existen múltiples técnicas como *Violation Transfer* o *Reliability branching* que no incluiremos en este trabajo y se pueden consultar en [10] y [1].

Finalmente, para la elección del punto de ramificación hay distintas técnicas que se pueden emplear, en función de la dificultad de resolución de los problemas resultantes. Una de ellas toma como punto x_i^b aquel que minimiza la suma de las áreas de las regiones factibles resultantes. Este

[1] El escalado por $1 + ||\nabla\vartheta_i(\bar{\boldsymbol{x}}^k)||_2$ evita tomar como variables de ramificación aquellas con $[x_i^l, x_i^u]$ pequeño, que podrían no mejorar la linealización.

punto, en el caso de tener $\vartheta_j : \mathbb{R} \to \mathbb{R}$, es el punto de $\vartheta_j(x_i)$ con mayor distancia a la recta que define la envoltura superior, y que podemos calcular como

$$x_i^b = \dot{\vartheta}_j^{-1} \left(\frac{\vartheta_j(x_i^u) - \vartheta_j(x_i^l)}{x_i^u - x_i^l} \right),$$

donde $\dot{\vartheta}_j = \frac{\partial \vartheta_j}{\partial x_i}$. Este procedimiento se ilustra en la Figura 2.3. En ella se muestra el proceso para ejemplo (EJ') con $x_6 = x_2^2$ y $x_2 \in [0, 4]$. En 2.3a vemos la solución $(\bar{x}_2, \bar{x}_6)$ que se encuentra dentro de la región factible linealizada pero $\bar{x}_6 > \bar{x}_2^2$, por lo que no se cumple la restricción original y es necesario ramificar (recordemos que x_2 no era una variable entera). El resultado de la ramificación se puede ver en 2.3b, donde el punto x_i^b se ha obtenido minimizando el área de las regiones resultantes. Se puede ver que efectivamente con cada paso del algoritmo la región relajada aproximará cada vez mejor la región no lineal original.

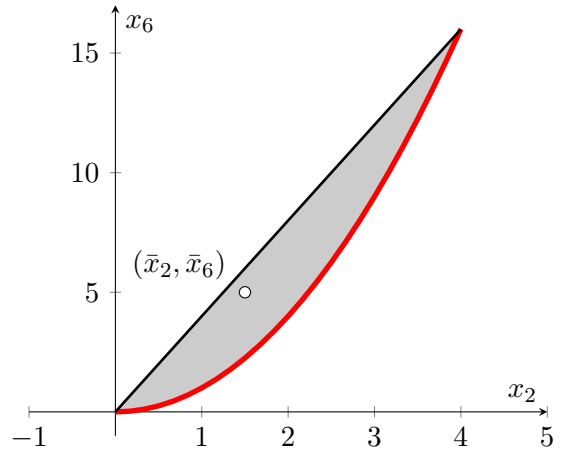
Tras este paso se crearían dos nuevos subproblemas añadiendo las restricciones $x_i \leq x_i^b$ y $x_i \geq x_i^b$, y se repetiría el proceso, linealizando cada uno de los subproblemas, calculando su solución y viendo si es necesario refinar la linealización o ramificar.

Como ejemplo de este nuevo paso se ha incluido la Figura 2.3c donde se puede ver la ramificación en P_2 y P_3 y la nueva linealización. Además, se nota que la región aproximada es cada vez más exacta, pues hay mucha menos diferencia entre las curvas representadas en rojo y la envoltura.

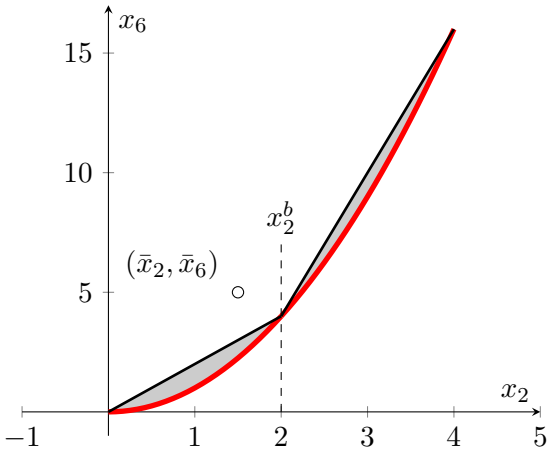
2.3. Recapitulando

Este capítulo nos ha introducido en la forma de resolver MINLPs siguiendo la aproximación empleada por COUENNE. Además de este optimizador, existen muchos otros de carácter global. Si tomamos otro habrá numerosas diferencias en la forma de realizar cada uno de los pasos básicos que hemos comentado: reformulación, linealización y ramificación. En consecuencia, parece evidente que los distintos optimizadores funcionarán mejor en unos problemas que en otros, dependiendo de las características del mismo. Dado un problema de optimización MINLP nos interesaría saber qué optimizador será el que funcionará mejor para ese problema. Ese será el objetivo del Capítulo 4 de este trabajo; dado un problema saber qué optimizador lo resolverá mejor.

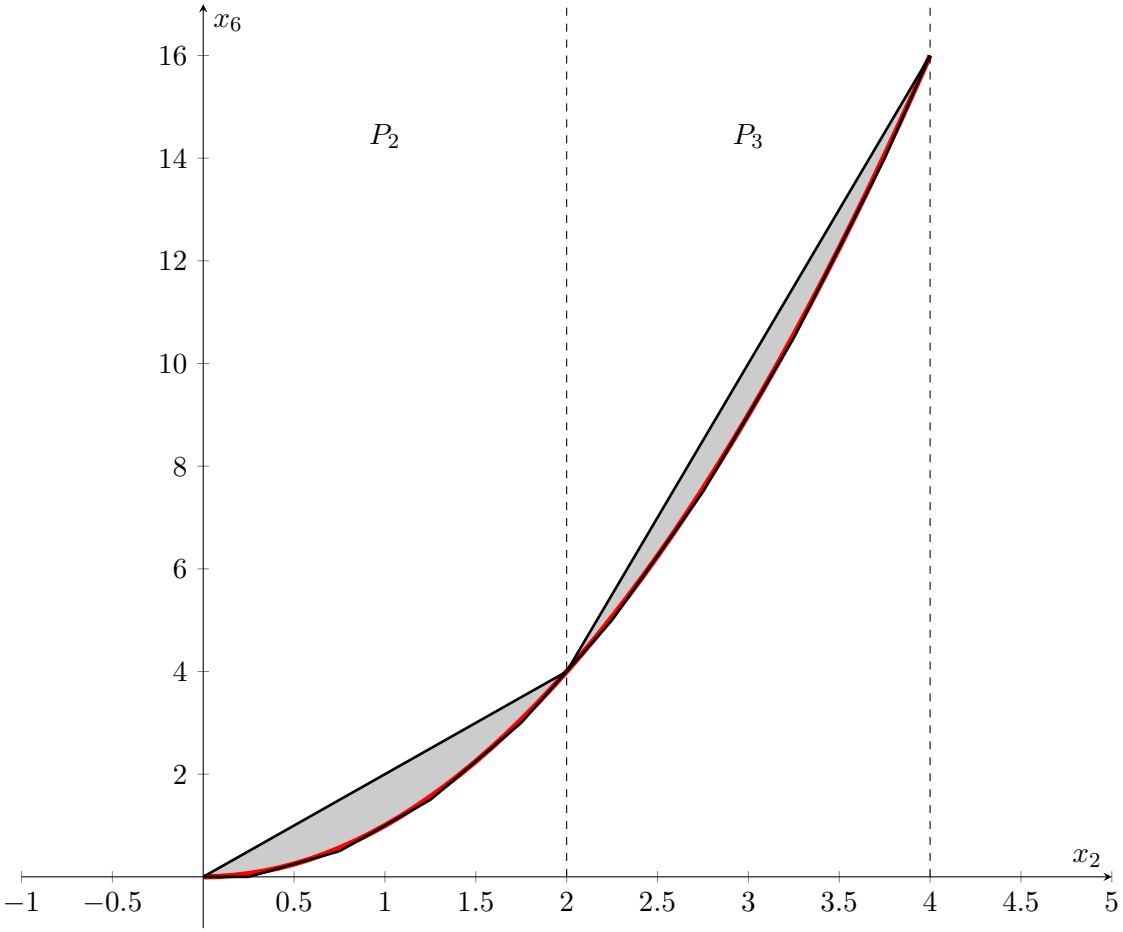
Antes de eso, presentaremos en el capítulo siguiente dos técnicas de aprendizaje conocidas, que se emplearán en el estudio que se llevará a cabo en el Capítulo 4.



(a) Solución infactible con la restricción $x_6 = x_2^2$.



(b) Resultado de la ramificación.



(c) Nueva linealización.

Figura 2.3: Procedimiento de ramificación para $x_6 = x_2^2$.

Capítulo 3

Revisión de algunas metodologías de aprendizaje estadístico

En el campo de la programación matemática, que abordamos en los capítulos anteriores, se han desarrollado numerosos optimizadores. La elección del optimizador adecuado para resolver un problema de optimización global puede ser crucial para obtener una solución en un tiempo razonable. Esto es especialmente relevante en el caso de la resolución de problemas "difíciles", los MINLPs. En estos casos, la selección del optimizador apropiado es aún más importante. Nuestro estudio se enfocará en determinar qué optimizador funciona mejor para un problema de programación matemática, basándonos en sus características.

Las bases teóricas del estudio se explicarán en este capítulo, mientras que la explicación del experimento y los resultados asociados los abordaremos en el capítulo siguiente.

Comenzaremos el capítulo con una introducción a lo que se conoce como aprendizaje estadístico, para luego comentar algunas de las técnicas más utilizadas y que emplearemos en el experimento computacional. Comenzaremos con una de las técnicas más sencillas y más empleadas, la regresión lineal múltiple (Sección 3.2), para terminar con una técnica más reciente que ha ido cobrando fuerza en los últimos años, las redes neuronales (Sección 3.3).

Para el desarrollo del capítulo se han consultado las secciones 2.3 y 11.3-11.5 de [6] junto con [2], donde se pueden ampliar los métodos incluidos en este trabajo.

3.1. Introducción al aprendizaje estadístico

El aprendizaje estadístico es una rama que combina la inteligencia artificial y la estadística centrada en el desarrollo de algoritmos y técnicas para "aprender de los datos". La idea esencial de estos algoritmos es crear modelos que recojan el conocimiento proporcionado por los datos y que sean capaces de generalizarlo, prediciendo valores para nuevos datos. En este campo es vital importancia tener una muestra de observaciones que servirán para entrenar un modelo.

Generalmente dispondremos de una o varias variables de interés (**variables respuesta**), que podrán ser cuantitativas o cualitativas y para las que querremos predecir su valor, en base a otro conjunto de variables que conforman las características del problema (**variables explicativas**).

Para realizar esta tarea tendremos una serie de observaciones de nuestras variables explicativas con su correspondiente variable respuesta. Mediante las técnicas que se introducirán en las secciones posteriores usaremos estas observaciones, que conformarán el conjunto de entrenamiento, para construir un modelo que sea capaz de predecir la salida para un nuevo conjunto de observaciones. Un buen modelo será aquel que sea capaz de realizar buenas predicciones sobre las variables respuesta.

Este tipo de método de aprendizaje se denomina **aprendizaje supervisado**, debido a que disponemos de la respuesta asociada al conjunto de variables explicativas y por lo tanto podemos "supervisar" el proceso de aprendizaje, comprobando la calidad del modelo desarrollado. Para llevar a cabo esta tarea se separa el conjunto de observaciones en dos subconjuntos: un conjunto de entrenamiento (o *training set*) y un conjunto de test (*test set*). El primero de ellos será el que se utilizará para crear y entrenar el modelo, mientras que el segundo servirá para validar su calidad, prediciendo el valor de las salidas para cada instancia de este. Habitualmente se selecciona un 70 % de la muestra como conjunto de entrenamiento, dejando el 30 % restante para el conjunto de test.

Según el tipo de problema de aprendizaje que tengamos podemos hablar de problemas de clasificación, en los que lo que se busca es dar una respuesta categórica, agrupando los problemas en clases o categorías, o de regresión, en el que se busca dar una respuesta cuantitativa y, en muchos casos, continua. Este último tipo será el que nos interesará, pues definiremos una medida de rendimiento o KPI (*Key Performance Indicator*) para los problemas resueltos e intentaremos predecir cual es el optimizador que mejor funciona en base a esta medida numérica.

Siguiendo las notaciones utilizadas en [6] en un problema de regresión denotaremos por X a la variable explicativa, pudiendo ser vectorial, $X = (X_1, \ldots, X_P)$, en el caso de disponer de P-variables explicativas. Por su parte, la variable o variables respuesta estarán denotadas por Y o $Y = (Y_1, \ldots, Y_K)$. Emplearemos minúsculas para hablar de las observaciones i-ésimas de una

variable explicativa, x_i, que de nuevo podrán ser vectoriales $x_i = (x_{i1}, \ldots, x_{iN})$. Representaremos las matrices en negrita, por ejemplo si tenemos un conjunto de N observaciones de P variables $x_i, i = 1, \ldots, N$ tendremos $\boldsymbol{X} \in Mat_{N\times P}(\mathbb{R})$. Finalmente, dado un vector de características X denotaremos por $\hat{Y}$ a las predicciones sobre Y. Tendremos entonces un conjunto de observaciones $(x_i, y_i), i = 1, \ldots, N$, sobre el cual construiremos los modelos.

3.2. Regresión lineal múltiple: modelo lineal general

El modelo lineal ha sido uno de los modelos más extendidos y conocidos durante los últimos 40 años. Dado un vector de características de entrada $X^T = (X_1, \ldots, X_P)$ el modelo lineal predice el valor de Y según

$$\hat{Y} = \hat{\beta}_0 + \sum_{p=1}^{P} \hat{\beta}_j X_j,$$

donde el término $\hat{\beta}_0$ se denomina intercepto o *bias*, en el ámbito del *machine learning*.

El modelo anterior se puede escribir como un producto matricial, si incluimos 1 como primera componente del vector de características $X = (1, X_1, \ldots, X_P)$ y $\hat{\beta}_0$ en $\hat{\beta} = (\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_P)$ como $\hat{Y} = X^T\hat{\beta}$, donde X^T denota el vector traspuesto o matriz traspuesta de X.

Para ajustar el modelo anterior hay diversos métodos, pero el más empleado es la estimación por mínimos cuadrados. Consiste en tomar los coeficientes β tales que se minimice la suma residual de cuadrados

$$RSS(\beta) = \sum_{i=1}^{N} (y_i - x_i^T\beta),$$

que se puede escribir en notación matricial,

$$RSS(\beta) = (Y - \boldsymbol{X}\beta)^T(Y - \boldsymbol{X}\beta),$$

donde $\boldsymbol{X}$ se denomina matriz de diseño de dimensión $N \times (P + 1)$ y en cada columna contiene el valor de la variable explicativa p-ésima para la observación i-ésima, menos la primera que es todo 1s. Y será el vector N-dimensional de salidas para el conjunto de entrenamiento.

$$\boldsymbol{X} = \begin{pmatrix} 1 & x_{11} & \cdots & x_{1P} \\ 1 & x_{21} & \cdots & x_{2P} \\ \vdots & \vdots & x_{ip} & \vdots \\ 1 & x_{N1} & \cdots & x_{NP} \end{pmatrix}_{N\times(P+1)}$$

Reescribiendo $RSS(\beta)$ teniendo que cuenta que $Y^T\boldsymbol{X}\beta = \beta^T\boldsymbol{X}^TY \in \mathbb{R}$ tenemos

$$\begin{aligned} RSS(\beta) =& (Y-\boldsymbol{X}\beta)^T(Y-\boldsymbol{X}\beta) = Y^TY - Y^T\boldsymbol{X}\beta - \beta^T\boldsymbol{X}^TY + \beta^T\boldsymbol{X}^T\boldsymbol{X}\beta \\ =& Y^TY - 2\beta^T\boldsymbol{X}^TY + \beta^T\boldsymbol{X}^T\boldsymbol{X}\beta, \end{aligned}$$

y derivando con respecto a β e igualando a 0 obtenemos lo que se conoce como **ecuaciones normales de la regresión**

$$\boldsymbol{X}^T(Y-\boldsymbol{X}\beta) = 0.$$

Si $\boldsymbol{X}^T\boldsymbol{X}$ es no singular, podemos estimar β como

$$\hat{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^TY,$$

obteniendo la siguiente predicción para para los valores del conjunto de entrenamiento

$$\hat{Y} = \boldsymbol{X}\hat{\beta} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^TY = \boldsymbol{H}Y.$$

Con esta estimación podemos calcular el valor de la predicción para el i-ésimo dato (o vector) de entrada como $\hat{y}_i = \hat{y}(x_i) = x_i^T\hat{\beta}$. La matriz $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T$ se denomina *matriz hat* y está relacionada con la interpretación geométrica de la estimación realizada.

Si denotamos por $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_P$ a las columnas de $\boldsymbol{X}$ tenemos que estos vectores formarán un subespacio de $\mathbb{R}^N$. Al minimizar $RSS(\beta)$ buscamos un estimador $\hat{\beta}$ tal que el vector de residuos $Y - \hat{Y}$ sea ortogonal a este subespacio de columnas de $\boldsymbol{X}$, y por lo tanto la estimación $\hat{Y}$ es la proyección ortogonal de Y sobre este subespacio. La matriz $\boldsymbol{H}$ es la que realiza esta proyección. Podemos ver una representación de esto en la Figura 3.1 en caso de tener dos variables explicativas $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$.

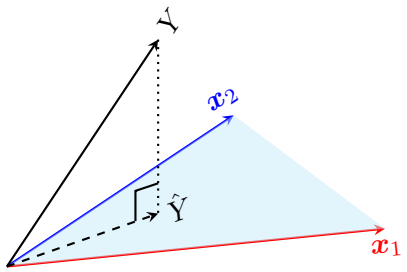


Figura 3.1: Proyección ortogonal de Y sobre el subespacio formado por $\boldsymbol{x}_1$ y $\boldsymbol{x}_2$.

Por último notar que hemos supuesto que la matriz $\boldsymbol{X}^T\boldsymbol{X}$ es no singular. En el caso de que las columnas de $\boldsymbol{X}$ sean linealmente independientes tendríamos que la matriz $\boldsymbol{X}^T\boldsymbol{X}$ sería una matriz cuadrada de orden $(P+1)$, simétrica y semidefinida positiva. Sin embargo, si las columnas de $\boldsymbol{X}$ no fuesen linealmente independientes habría variables correladas y la proyección

$\hat{Y}$ sobre el subespacio de columnas de $\boldsymbol{X}$ no estaría definida de manera única como combinación de las columnas de $\boldsymbol{X}$. En muchos casos esto se debe a una elección redundante de las variables explicativas y se podría solucionar redefiniendo alguna variable o eliminándola. Este proceso se hace de forma automática en la mayoría de librerías de aprendizaje estadístico, por lo que en la práctica no resulta un problema añadido.

3.3. Redes neuronales

En esta sección, exploraremos un método de aprendizaje supervisado ampliamente utilizado en los últimos años: el método de ajuste de redes neuronales mediante el algoritmo de retropropagación. Para ello nos basaremos en las secciones 11.3-11.5 de [6].

Las redes neuronales están diseñadas para emular el comportamiento del cerebro humano, de ahí su nombre, y son aplicables tanto a problemas de clasificación como de de regresión. Cada nodo en la red representa una neurona, y las conexiones entre ellas modelan las sinapsis. La estructura básica de una red neuronal comprende una capa de entrada, que recibe los datos (las variables predictoras), una o varias capas intermedias y una capa de salida, con las predicciones o clasificación. De esta manera, si tenemos un vector $X = (X_1, \ldots, X_P)$ de P características (variables predictoras), podemos utilizar la red para predecir un valor de salida (en el caso de regresión) o clasificar el problema en diferentes clases. Podemos ver una representación de una red neuronal en la Figura 3.2.

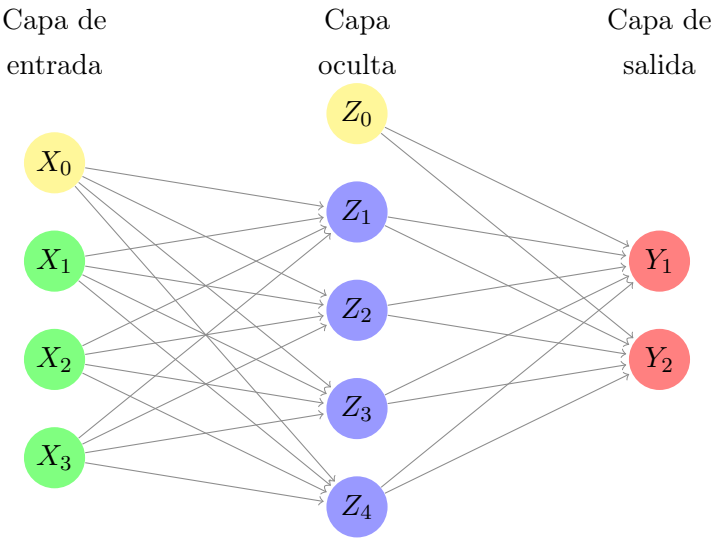


Figura 3.2: Esquema de una neuronal con una capa oculta.

En el esquema anterior tenemos el vector 3 neuronas en la capa de entrada (en verde),

$X = (X_1, X_2, X_3)$, una capa oculta dada por $Z = (Z_1, Z_2, Z_3, Z_4)$ (en azul) y una capa de salida $Y = (Y_1, Y_2)$ (en rojo). Habitualmente en los problemas de regresión la capa de salida tendrá una sola neurona, pero explicaremos el procedimiento para el caso general con P características de entrada, M neuronas en la capa intermedia y K neuronas en la de salida.

Por otra parte, en la figura también se han incluido en amarillo datos de entrada a las capas oculta y final sin conexiones entrantes, denominados sesgos o *bias*. Añadiendo una nueva neurona en las capas de entrada (X_0) y oculta (Z_0) con valor 1 podemos modelizar estos sesgos, obteniendo $X = (1, X_1, \ldots, X_P)$ y $Z = (1, Z_1, \ldots, Z_M)$.

En consecuencia, la red anterior se puede formular de manera general mediante

$$\begin{aligned} Z_m &= \sigma(\alpha_m^T X),\ m = 1, \ldots, M \\ T_k &= \beta_k^T Z,\ k = 1, \ldots, K \\ Y_k &= f_k(X) = g_k(T),\ k = 1, \ldots, K. \end{aligned} \tag{3.1}$$

donde σ se denomina *función de activación* y normalmente se trata de la función sigmoide.[1] La introducción de esta función permite aumentar el número de modelos representados mediante las redes, pues en el caso de tomar la identidad el modelo final resultaría en un modelo lineal sobre las variables de entrada. Además, las funciones g_k permiten una última transformación de los datos de salida y en problemas de regresión suelen tomarse la identidad.

Para que la red neuronal sea funcional hay que ajustar una serie de parámetros, que denominamos pesos. El conjunto de todos los pesos lo denotaremos por θ, que esencialmente consiste en

$$\begin{aligned} \alpha_m &= (\alpha_{0m}, \ldots, \alpha_{pm}, \ldots, \alpha_{Pm}) \in \mathbb{R}^{P+1},\ m = 1, \ldots, M \\ \beta_k &= (\beta_{0k}, \ldots, \beta_{mk}, \ldots, \beta_{Mk}) \in \mathbb{R}^{M+1},\ k = 1, \ldots, K, \end{aligned}$$

o más en detalle,

$$\begin{aligned} \alpha &= \left(\alpha_1 \,\middle|\, \cdots \,\middle|\, \alpha_m \,\middle|\, \cdots \,\middle|\, \alpha_M \right) = \begin{pmatrix} \alpha_{01} & \alpha_{02} & \cdots & \alpha_{0M} \\ \alpha_{11} & \alpha_{12} & \cdots & \alpha_{1M} \\ \vdots & \vdots & \alpha_{pm} & \vdots \\ \alpha_{P1} & \alpha_{12} & \cdots & \alpha_{PM} \end{pmatrix}_{(P+1)\times M} \\ \beta &= \left(\beta_1 \,\middle|\, \cdots \,\middle|\, \beta_k \,\middle|\, \cdots \,\middle|\, \beta_K \right) = \begin{pmatrix} \beta_{01} & \beta_{02} & \cdots & \beta_{0K} \\ \beta_{11} & \beta_{12} & \cdots & \beta_{1K} \\ \vdots & \vdots & \beta_{mk} & \vdots \\ \beta_{M1} & \beta_{12} & \cdots & \beta_{MK} \end{pmatrix}_{(M+1)\times K} \end{aligned} \tag{3.2}$$

[1] $\sigma(v) = \frac{1}{1+e^{-v}}$

donde cada elemento α_{pm} de α se corresponde con el peso del arco $X_p \to Z_m$ y cada β_{mk} de β con el peso del arco $Z_m \to Y_k$. Denotaremos como α_m a la columna m-ésima de α que se corresponde con los pesos de entrada para cada neurona de la capa intermedia Z_m y β_k a la columna k-ésima de β correspondiente a los pesos de los arcos de entrada de la neurona de salida Y_k.

El ajuste de la red se realizará buscando minimizar una función de error $R(\theta)$ variando el conjunto de pesos θ, que para el caso de la regresión será

$$R(\theta) = \sum_{i=1}^{N} R_i = \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \tag{3.3}$$

donde y_{ik} es la coordenada k-ésima de la salida para los datos de entrada x_i y $f_k(x_i)$ es el valor devuelto por la red neuronal. El ajuste de estos parámetros se hace mediante el descenso del gradiente, mediante el método de retropropagación. Este método recorre la red neuronal desde la entrada hasta la salida, computando los valores de cada neurona y calculando el error cometido. Posteriormente se reajustan los pesos hacia atrás, repitiendo el procedimiento un número de veces hasta que se consigue el mínimo.

Sea entonces $z_{im} = \sigma(\alpha_m^T x_i)$ el valor de la neurona Z_m para los datos de entrada i-ésimos y $z_i = (z_{i1}, z_{i2}, \ldots, z_{iM})$ el vector de valores de las neuronas de la capa oculta para la entrada x_i. Nuestro objetivo es ver como varía el error dado por (3.3) al variar los pesos de la red neuronal. Así, si variamos el peso β_{mk} asociado al arco que va desde Z_m a Y_k para los datos de entrada x_i tendremos lo siguiente, aplicando la regla de la cadena convenientemente en (3.3)

$$\frac{\partial R_i}{\partial \beta_{mk}} = \frac{\partial R_i}{\partial f_k} g_k'(\beta_k^T z_i) \frac{\partial (\beta_k^T z_i)}{\partial \beta_{mk}} = -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{im}. \tag{3.4}$$

De igual manera, si variamos ahora el peso α_{pm}, asociado al arco que va desde la neurona de entrada X_p a la de la capa intermedia Z_m tenemos

$$\begin{aligned} \frac{\partial R_i}{\partial \alpha_{pm}} &= \sum_{k=1}^{K} \frac{\partial R_i}{\partial f_k} g_k'(\beta_k^T z_i) \frac{\partial (\beta_k^T z_i)}{\partial z_{im}} \sigma'(\alpha_m^T x_i) \frac{\partial (\alpha_m^T x_i)}{\partial \alpha_{pm}} \\ &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{mk} \sigma'(\alpha_m^T x_i) x_{ip}. \end{aligned} \tag{3.5}$$

Notar que en este caso el cambio sobre el peso α_{pm} afecta a todas las neuronas de salida que dependan de la neurona Z_m, de ahí el sumatorio en la fórmula anterior.

Podemos escribir las anteriores ecuaciones como $\frac{\partial R_i}{\partial \beta_{mk}} = \delta_{ik} z_{im}$ y $\frac{\partial R_i}{\partial \alpha_{pm}} = s_{im} x_{ip}$ que nos proporcionan el error del modelo actual en las capas de salida y oculta, respectivamente. Usando estas fórmulas podemos relacionar (3.4) y (3.5) mediante

$$s_{im} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{mk} \delta_{ik} \tag{3.6}$$

conocidas como **ecuaciones de retropropagación**.

Empleando las ecuaciones (3.4) y (3.5), el método de descenso del gradiente en la iteración $(r+1)$-ésima tomaría la forma

$$\begin{aligned} \beta_{mk}^{(r+1)} &= \beta_{mk}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{mk}^{(r)}}, \\ \alpha_{pm}^{(r+1)} &= \alpha_{pm}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{pm}^{(r)}}, \end{aligned} \tag{3.7}$$

donde γ_r es la tasa de aprendizaje que regulará la velocidad de aprendizaje de la red.

Usando (3.6) las modificaciones sobre α_{pm} y β_{mk} se realizarán en dos pasos. En el *paso hacia adelante* se fijan unos pesos y se calculan los las predicciones $\hat{y}_{ik} = f_k(x_i)$ mediante (3.1). En el *paso hacia atrás* se calculan los errores δ_{ik} de las predicciones anteriores y se "retropropagan" mediante (3.6) para conseguir los s_{im}. Con los errores calculados se pueden modificar los pesos según (3.7) y repetir el proceso. Este algoritmo en dos pasos es lo que se conoce como **algoritmo de retropropagación**.

En muchos casos las redes neuronales crean modelos sobreajustados de los datos de entrenamiento, realizando malas predicciones para datos nuevos, por ejemplo, en el conjunto de test. Una manera de evitar este sobreajuste del modelo es la técnica de decaimiento de pesos (*weight decay* en inglés). Esta técnica consiste en añadir a la función de error una función de penalización obteniendo una nueva función de error $R(\theta) + \lambda J(\theta)$, donde

$$J(\theta) = \sum_{mk} \beta_{mk}^2 + \sum_{pm} \alpha_{pm}^2,$$

con $\lambda \geq 0$. Con un valor alto de λ los pesos β_{mk} y α_{pm} se irían haciendo más pequeños con el avance en el ajuste, para minimizar los términos $2\beta_{mk}$ y $2\alpha_{pm}$ que ahora aparecerían en las expresiones (3.4) y (3.5).

Por último, quedan pendientes la elección de la tasa de aprendizaje γ_r y los pesos iniciales. En cuanto a la tasa de aprendizaje normalmente se elige constante $\gamma_r = \gamma, \forall r$ pero se podría ir modificando para minimizar el error en cada iteración. Los valores iniciales de los pesos $\alpha^{(0)}$ y $\beta^{(0)}$ se suelen tomar cercanos a 0 de forma aleatoria, de forma que el modelo empezaría a operar de forma casi lineal e iría convirtiéndose en no lineal a medida que avanza el algoritmo. Valores altos de los pesos iniciales no suelen conseguir buenos resultados.

Con la técnica que acabamos de comentar tenemos todos los ingredientes para presentar el problema de aprendizaje de interés y obtener resultados numéricos preliminares del siguiente capítulo.

Capítulo 4

Aprendizaje para la selección de algoritmos de optimización global

Como ya hemos comentado anteriormente, el objetivo final de este trabajo es realizar una comparativa de distintos optimizadores globales en el campo de la optimización no lineal. Tras un recordatorio de la programación lineal entera mixta en el Capítulo 1 y la explicación del método de Ramificación y Acotación Espacial para la resolución de MINLPs en el Capítulo 2, será en este capítulo en el que presentemos el experimento computacional y expondremos los resultados obtenidos. Para ello se emplearán las técnicas de aprendizaje estadístico explicadas en el Capítulo 3.

Para realizar las comparaciones entre los distintos optimizadores, utilizaremos una librería de R desarrollada por CITMAga, en la cual tuve la oportunidad de colaborar durante unas prácticas de verano. El objetivo original de esta librería era comparar el desempeño del optimizador polinómico RAPOSa con distintas configuraciones, obteniendo un modelo que predijese la mejor configuración para cada problema. Este caso de estudio no será en el que nos centremos, sino en la selección del mejor optimizador para resolver un problema dado. Sin embargo, la librería está pensada para realizar tareas de aprendizaje estadístico generales y será de utilidad para llevar a cabo nuestro objetivo.

Como en cualquier proceso de aprendizaje estadístico será fundamental la selección de los datos que se emplearán para entrenar el modelo y realizar las predicciones. Es por esto que incluiremos una primera sección donde explicaremos la batería de problemas utilizados y las características de estos. Posteriormente se explicará la medida de rendimiento que se va a emplear para el aprendizaje para finalizar con la implementación en R de los dos métodos del Capítulo 3 y la discusión de los resultados obtenidos.

Para el desarrollo de este capítulo experimental se ha seguido una metodología similar al Capítulo 3 de [2], pero adaptándola al problema propio y las técnicas seleccionadas.

4.1. Selección del conjunto de datos

En esta sección explicaremos el conjunto de datos empleados para realizar las ejecuciones y llevar a cabo el proceso de aprendizaje. Nuestro conjunto de datos se ha extraído de la librería MINLPLib (*A Library of Mixed-Integer and Continuous Nonlinear Programming Instances*) [11], que proporciona más de 1600 instancias de problemas MINLP y NLP. Esta librería incluye cada problema en varios formatos para poder ser resueltos, por ejemplo, mediante AMPL.

Uno de los primeros pasos que debemos realizar será seleccionar el conjunto de variables explicativas X que servirán para realizar el aprendizaje. Seleccionaremos entre las variables numéricas aquellas más significativas, que se detallan a continuación:

- `nvars`: número de variables del problema.
- `nbinvars`: número de variables binarias.
- `nintvars`: número de variables enteras (no binarias).
- `nnlvars`: número de variables que aparecen de forma no lineal en la función objetivo o funciones de restricción.
- `nnlbinvars`: número de variables binarias que aparecen de forma no lineal en la función objetivo o funciones de restricción.
- `nnlintvars`: número de variables enteras que aparecen de forma no lineal en la función objetivo o funciones de restricción.
- `ncons`: número de restricciones.
- `nlincons`: número de restricciones lineales no constantes.
- `nquadcons`: número de restricciones cuadráticas no lineales.
- `npolynomcons`: número de restricciones polinómicas no cuadráticas
- `nsignomcons`: número de restricciones signomiales no polinómicas.
- `ngennlcons`: número de restricciones genéricas no signomiales.
- `njacobiannz`: suma del número de variables que aparecen en cada función de restricción.

- `njacobiannlnz`: suma del número de variables que aparecen de forma no lineal en cada función de restricción.
- `mincoef`: valor absoluto más pequeño de los coeficientes no nulos de las funciones objetivo y de restricción. En caso de tener restricciones no lineales se miran también las constantes numéricas que aparecen en ellas.
- `maxcoef`: valor absoluto más grande de los coeficientes no nulos de las funciones objetivo y de restricción. En caso de tener restricciones no lineales se miran también las constantes numéricas que aparecen en ellas.

Por lo tanto, dispondremos de 16 variables explicativas, relacionadas con el número de variables del problema y sus tipos, el número de restricciones y los coeficientes máximo y mínimo.

Para realizar las ejecuciones tomaremos 1161 problemas de la librería MINLPLib y los resolveremos con Octeract, Lindoglobal, Baron y Couenne. Para cada problema y cada optimizador guardaremos el *tiempo* de ejecución, los valores finales de UB y LB y el *gap*, que se puede calcular como $gap = UB - LB$. Con esta información podremos definir la medida de rendimiento o KPI.

Realizando un análisis de los datos salidos de las ejecuciones vimos que había ciertas discrepancias en cómo unos optimizadores u otros devolvían algunos de los valores que nos servirán para realizar el aprendizaje. En ciertos casos hay optimizadores que dan por resuelto un problema, devolviendo unos valores de UB y LB muy dispares a los que devuelven sus compañeros. Como esto podría influir negativamente en el proceso de aprendizaje, se eliminarán los problemas que producen este tipo de errores para algún optimizador, resultando en 836 problemas finalmente.

Como en cualquier proceso de aprendizaje supervisado se dividirá el conjunto de datos en dos subconjuntos. Un 70 % de los problemas iniciales (585 problemas) conformará el conjunto de entrenamiento, que servirá para crear y entrenar el modelo, y el 30 % restante formará el conjunto de test (349 problemas), para poder validar el modelo creado.

Una vez seleccionado el conjunto de problemas que van a formar parte del aprendizaje, el siguiente paso será definir una medida de rendimiento que nos permitirá comparar el desempeño de los distintos optimizadores. Esta medida se calculará en base a la información recuperada de las ejecuciones realizadas.

4.2. Obtención de la variable respuesta. Cálculo del KPI

Para la obtención de la solución mediante los optimizadores comentados anteriormente se fijó un tiempo máximo de ejecución de 3600 segundos y una tolerancia de $tol = 0.001$. Así, se

considerará que un problema se ha resuelto si $|UB - LB| < tol$. Para algún problema que no se diese esta condición la ejecución pararía si se superasen los 3600 segundos. Definiremos ahora la medida de rendimiento o KPI.

Parece natural pensar en tomar un KPI que dependiese únicamente del *tiempo* de ejecución o del *gap* devuelto. Una medida con estas características aportaría poca información. Si nos encontramos en el caso en el que un optimizador resuelve un problema con un $gap \approx 0$ pero rozando el *tiempo* límite, tener únicamente en cuenta el *gap* para calcular nuestro KPI resultaría en una resolución muy eficiente, cuando en la realidad no lo es. Si por el contrario tuviésemos solo en cuenta el *tiempo* de resolución, un optimizador que resuelva un problema en un $tiempo \approx 0$ pero devolviendo un *gap* grande sería una buena opción, cuando seguramente la solución obtenida no sea muy buena. Es por esto que se hace necesario un KPI que combine el *tiempo* y el *gap*.

Definiremos el KPI para nuestro problema como

$$KPI = (1 + tiempo)\left(1 + \frac{|UB - LB|}{UB + \varepsilon}\right),$$

donde ε toma un valor pequeño para evitar divisiones por 0. Podemos ver que se hace una combinación del *tiempo* y el *gap*, calculado de forma relativa respecto del valor de UB. Con esta formulación un optimizador será muy bueno si resuelve el problema en un *tiempo* muy bajo y con un *gap* muy pequeño, obteniendo por lo tanto un valor de KPI muy cercano a 1. Si un optimizador consume todo el tiempo tendrá un KPI asociado mayor que 3600, y la calidad dependerá del *gap* devuelto.

Esta medida de rendimiento se calculará para cada problema i, con $i = 1, \ldots, 836$, y cada optimizador j, con $j = 1, \ldots, 4$, obteniendo una matriz $KPI \in \mathbb{R}^{836\times 4}$. El siguiente paso es entrenar un modelo para cada optimizador, con el conjunto de variables explicativas de entrenamiento y su KPI asociado. Sin embargo, como nuestro objetivo es comparar los optimizadores, encontrando aquel que produce mejores resultados para un problema dado, este entrenamiento por sí solo no sería suficiente, ya que generaría modelos que no guardarían relación. Esto hace necesario un paso adicional. Partiendo de nuestra matriz de KPI

$$KPI = \begin{pmatrix} KPI_{1,1} & \cdots & KPI_{1,4} \\ \vdots & \ddots & \vdots \\ KPI_{836,1} & \cdots & KPI_{836,4} \end{pmatrix}_{836\times 4},$$

donde cada fila representa un problema y cada columna su KPI para un optimizador, tomaremos como variable respuesta de nuestro problema y_{ij}, dada como

$$y_{ij} = \frac{\min_k\{KPI_{ik}\}}{KPI_{ij}}, \ \forall i, j.$$

Con este cálculo tendremos que el mejor optimizador j para un problema i tendrá $y_{ij} = 1$ y el resto de optimizadores valores menores, todos ellos en $[0, 1]$. Esta transformación sobre la

variable respuesta del problema, además de crear una relación entre los optimizadores, facilitará la representación de los datos, al estar todos ellos en el mismo intervalo.

4.3. Implementación en R

Como se comentó al inicio del capítulo se empleará el lenguaje estadístico R para la realización del aprendizaje. En concreto, se empleará la librería *ASML (Algorithm Selection for Machine Learning*) desarrollada en el CITMAga. Esta librería, con la ayuda del paquete `caret` [8], proporciona una serie de funciones para procesar los datos de entrada, crear los conjuntos de entrenamiento y test, entrenar los modelos, hacer predicciones y mostrar los resultados en gráficos y tablas.

Antes de entrar en el aprendizaje, vamos a analizar como se comporta cada optimizador para el conjunto total de problemas (836 problemas), obteniendo la media geométrica y aritmética de la variable respuesta. Esto se puede ver en la Tabla 4.1, en la que Baron es el optimizador que mejores resultados produce en general, seguido de Octeract y Couenne, y dejando en último lugar a Lindoglobal.

	Media geométrica	**Media aritmética**
OCTERACT	0.3467	0.6602
LINDOGLOBAL	0.1734	0.5387
BARON	0.6547	0.8416
COUENNE	0.2327	0.6385

Tabla 4.1: Medias aritmética y geométrica de la variable respuesta para cada optimizador.

El objetivo del aprendizaje consistirá en ver si es mejor tomar en cada caso el optimizador propuesto por el modelo frente a escoger siempre Baron, que es el que mejor desempeño produce en general. Esto lo analizaremos mediante la mejora promedio que podemos conseguir sobre la variable respuesta tras realizar el aprendizaje con cada una de las técnicas. Dado que los resultados pueden depender significativamente de cómo se divida el conjunto de datos en subconjuntos de entrenamiento y test, tomaremos 8 muestras diferentes y observaremos los resultados promedio para obtener una evaluación más robusta. En la Tabla 4.2 se pueden ver los valores de partida para el conjunto de test de cada una de las 8 muestras.

Muestra	1	2	3	4	5	6	7	8
Mejor salida	0.8415	0.8451	0.8235	0.8345	0.8308	0.8299	0.8189	0.8383
Óptimo alcanzable	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000

Tabla 4.2: Comparación entre el mejor optimizador promedio y la salida óptima que se podría llegar a tener para el conjunto de test de cada una de las 8 muestras.

En la fila "Mejor salida" se incluye para cada muestra el valor máximo de los promedios de la salida que proporciona cada optimizador, es decir, el valor de medio de Y que tendríamos si tomásemos para cada muestra el optimizador promedio mejor. En la fila "Óptimo alcanzable" se incluye el mejor valor medio que se podría conseguir, que evidentemente es 1, si tomásemos para cada problema el mejor optimizador.

Estos datos nos permiten calcular el margen de mejora que podríamos conseguir mediante el aprendizaje, que se detalla en la Tabla 4.3, siendo de 0.1672.

Promedio mejor salida	**Promedio óptimo alcanzable**	**Margen de mejora**
0.8328	1.000	0.1672

Tabla 4.3: Margen de mejora que se puede conseguir mediante aprendizaje estadístico.

Una vez establecido el punto de partida podemos proceder a realizar el aprendizaje estadístico. El proceso de aprendizaje seguirá el siguiente esquema. Primero, se dividirá el conjunto de datos en 8 muestras, cada una de ellas con su conjunto de entrenamiento y de test asociado. Para cada muestra y cada optimizador se ajustará un modelo, empleando el conjunto de entrenamiento $y_{i1}, \ldots, y_{i4}$ con $i = 1, \ldots, 585$.

A continuación, con cada modelo se obtendrán las predicciones $\hat{Y}$ para el conjunto de problemas de test. Estas predicciones debemos compararlas con la salida real Y que obtuvimos al hacer las ejecuciones. Dado un problema i del conjunto de test con $i = 1, \ldots, 349$ se escogerá la mejor predicción $\hat{y}_i$ que será aquella más cercana a 1, es decir,

$$\hat{y}_i = \max_{j=1,\ldots,4} \hat{y}_{ij},$$

que tendrá un optimizador j asociado. Este optimizador sería el que idealmente resolvería mejor el problema, según el modelo que hemos creado.

Para verificar si el optimizador elegido es el que mejor resuelve el problema en la realidad, recuperaremos el valor real de la salida para ese optimizador j, es decir, el valor de y_{ij}. Si el modelo ha realizado la predicción correctamente el valor y_{ij} debería ser 1, pues estaríamos diciendo que

el optimizador elegido por el proceso de aprendizaje para el problema i es el asociado a j, que realmente era el que mejor se comportaba para ese problema. Se calculará el promedio de estos valores, comparándolo con el valor "Mejor salida" de la Tabla 4.2, con el objetivo de ver si el proceso de aprendizaje merece la pena, frente a escoger por defecto un optimizador en concreto de manera general.

Estas comparaciones se harán para los dos métodos de aprendizaje estudiados en el Capítulo 3, comenzando por la regresión lineal.

4.3.1. Aprendizaje mediante regresión lineal

En esta sección realizaremos el aprendizaje mediante regresión lineal, empleando el método `lm` de R. Recordemos que nuestro conjunto de datos se va a dividir en 8 muestras, cada una de ellas con sus conjuntos de entrenamiento y test asociados. Empleando la librería, separamos los datos en dos conjuntos para cada una de las muestras y ajustamos el modelo. Posteriormente realizamos las predicciones para el conjunto de test.

Como se explicó anteriormente, recuperaremos el valor real asociado a la mejor predicción para cada problema y realizaremos la media aritmética de estos valores, que etiquetaremos como "ML", para cada una de las 8 muestras. Esto se puede ver en la Tabla 4.4. Además, se incluye también la fila "Mejora", que muestra la mejora entre tomar el mismo optimizador para todas las instancias o aquel seleccionado mediante el aprendizaje para cada problema.

Muestra	1	2	3	4	5	6	7	8
Mejor salida	0.8415	0.8451	0.8235	0.8345	0.8308	0.8299	0.8189	0.8383
ML	0.8658	0.8823	0.8416	0.8543	0.8535	0.8525	0.8391	0.8685
Mejora	0.0243	0.0372	0.0181	0.0198	0.0227	0.0226	0.0202	0.0302

Tabla 4.4: Mejora entre escoger el optimizador promedio óptimo y escoger para cada problema el mejor optimizador según el aprendizaje con regresión lineal, para el conjunto de test.

Podemos ver que la mejora es significativa para casi todas las muestras del conjunto de test. Si ahora realizamos la media aritmética de las medidas anteriores obtenemos los resultados que se muestran en la Tabla 4.5

Promedio mejor salida	**Promedio ML**	**Mejora**
0.8328	0.8572	0.0244

Tabla 4.5: Mejora obtenida al realizar el aprendizaje con regresión lineal

donde vemos que se ha conseguido una mejora en la variable respuesta de 0.0244. Esta mejora podría parecer escasa, pero si la comparamos con el margen de mejora que podíamos obtener tenemos

$$\%_{mejorado} = \frac{0.0244}{0.1672} = 14.59\,\%,$$

lo que nos dice que hemos mejorado casi un 15 % de lo que podíamos mejorar.

Los resultados obtenidos los podemos ver también de forma gráfica, con funciones proporcionadas por la librería. Incluiremos los gráficos para la muestra 2, que es la que mejor resultado ha obtenido con el aprendizaje en términos de mejora sobre la variable respuesta. Comenzamos incluyendo un diagrama de cajas en la Figura 4.1. En el gráfico se incluye una caja para cada

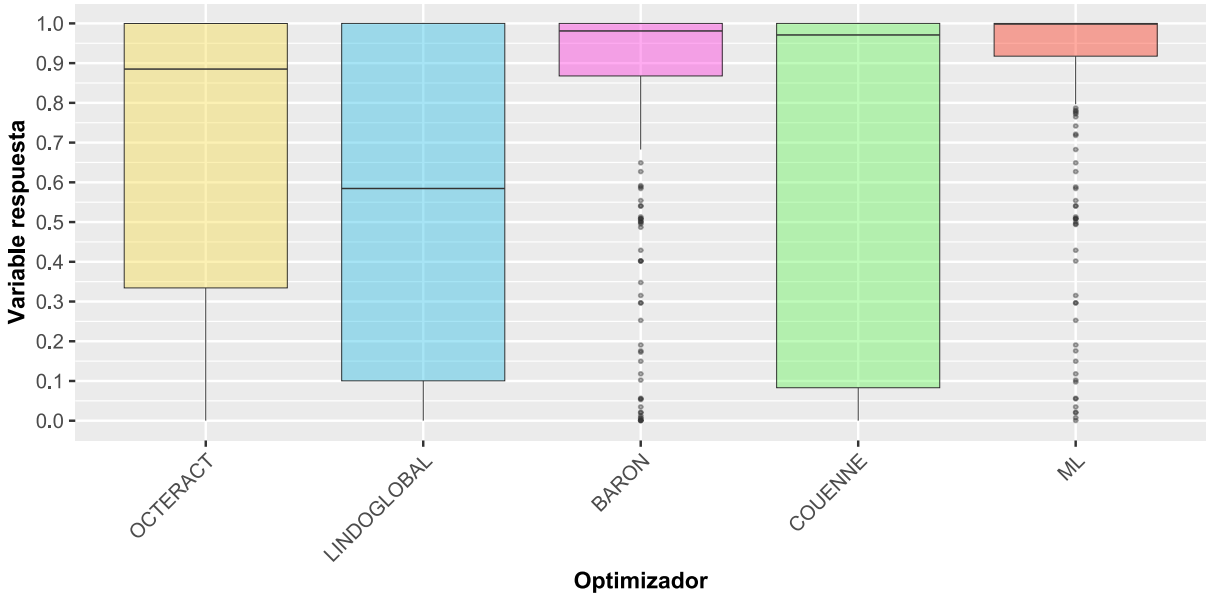
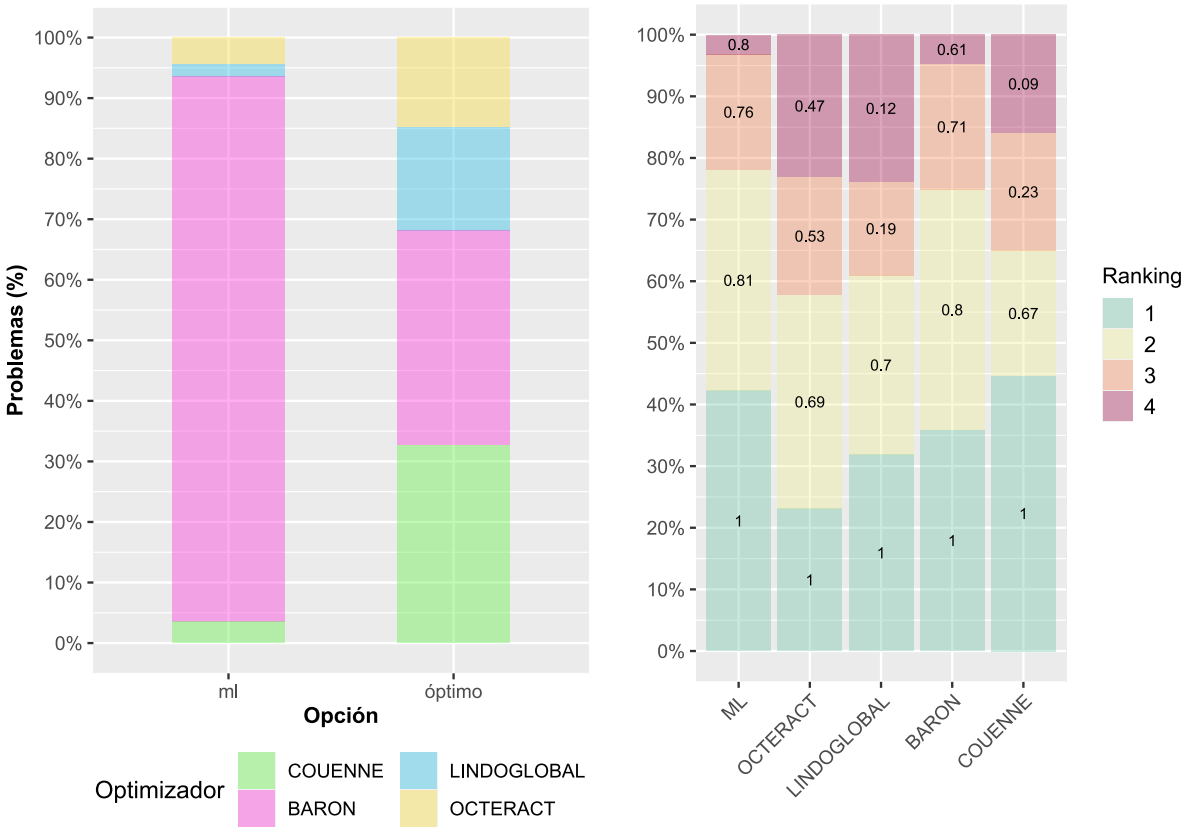


Figura 4.1: Diagramas de cajas para el aprendizaje con regresión lineal.

optimizador y una última caja etiquetada con "ML" para resumir el proceso de aprendizaje. Las cajas correspondientes a los optimizadores incluyen la representación de los valores de la variable respuesta Y para los problemas del conjunto de test. La caja "ML" representa los valores reales de la salida del optimizador escogido por el aprendizaje (aquel con mejor valor para la predicción) para cada problema. Podemos ver que la caja asociada al aprendizaje es más pequeña que la de Baron, el optimizador que mejor funciona en general, con lo que el aprendizaje proporciona mejores valores de la variable respuesta que tomar Baron siempre.

Por otra parte, se incluyen en la Figura 4.2 dos gráficos muy interesantes que se tienen que analizar conjuntamente. En el gráfico 4.2a tenemos una comparativa entre el porcentaje de problemas para los que cada optimizador es el óptimo (en la barra etiquetada como "óptimo") y

el porcentaje de problemas para los que el aprendizaje dice que cada optimizador es el óptimo (en la barra etiquetada como "ml"). Idealmente estas dos barras deberían ser iguales, pues entonces el modelo escogería siempre el optimizador óptimo para cada problema. Sin embargo, parece que el modelo toma en muchos más casos Baron como optimizador de referencia de lo que debería. Este razonamiento tiene su sentido, pues Baron es la herramienta que mejor se comporta en general y resolver un problema con ella nos debería de proporcionar un rendimiento bueno, aunque no fuese el mejor. Si ahora pasamos al gráfico 4.2b podremos ver el porcentaje de veces que cada



(a) Ranking de optimizadores y la variable respuesta asociada.

(b) Comparación entre el aprendizaje y el óptimo real.

Figura 4.2: Gráficos de comparación entre estrategias.

optimizador queda en cada posición, pudiendo haber empates, en el caso de que varios obtengan la solución con el mismo rendimiento. Así, COUENNE sería el optimizador que mejor resuelve los problemas un 45 % de las veces, Baron un 36 %, etc. Los números sobre cada barra apilada representan la media de la variable respuesta devuelta por los optimizadores cuando quedan en la posición de la barra. Aunque podría parecer que tomar COUENNE como opción de referencia es una buena estrategia general, hay que analizar más en detalle lo que ocurre con estos números.

Si tomamos COUENNE siempre, en el caso de ser el optimizador mejor tendríamos asegurado una variable respuesta de 1, mientras que si era el segundo, este valor cae a 0.67. Si nuestra elección es equivocada y COUENNE es el tercero o cuarto optimizador, estaríamos obteniendo unos valores promedio de rendimiento de 0.23 y 0.09 respectivamente, muy alejados del 1 que esperaríamos obtener. No obstante, tomando Baron, como sugería el gráfico anterior, aún cuando este optimizador fuese el último de los 4, y entonces el que peor resuelve el problema, estaríamos obteniendo un valor medio de 0.61, nada que ver con el 0.09 de antes. Es por esto que no resulta tan mala la "equivocación" al elegir Baron un número de veces mayor al óptimo. Finalmente, si nos fijamos en los valores sobre las barras y el tamaño de estas para el aprendizaje, etiquetado de nuevo como "ml", vemos que el modelo es bueno; se posiciona en primer lugar casi un 43 % de las veces y los valores asociados cuando esta posición decae son muy altos, los mayores de toda la gráfica.

En resumen, podemos decir que el aprendizaje con regresión lineal ha proporcionado mejoras frente a elegir por defecto un optimizador de manera general. Veremos en la subsección siguiente con las redes neuronales si conseguimos mejorar los resultados obtenidos.

4.3.2. Aprendizaje mediante redes neuronales de una capa oculta

En esta sección el aprendizaje se hará con el método `nnet` de R. De nuevo, se entrenará un modelo para cada optimizador y cada muestra, haciendo las predicciones para el conjunto de test. Recordemos que estamos realizando el aprendizaje con una red neuronal con una sola capa oculta, habiendo otros muchos métodos que involucran a las redes neuronales más complejos. El aprendizaje se hará con redes neuronales con 50 neuronas en la capa oculta y un valor de $\lambda = 0.1$, pues es con la combinación de parámetros con los que mejores resultados se han obtenido. En la Tabla 4.6 tenemos los resultados del aprendizaje para cada muestra, de manera análoga al caso de la regresión lineal.

Muestra	1	2	3	4	5	6	7	8
Mejor salida	0.8415	0.8451	0.8235	0.8345	0.8308	0.8299	0.8189	0.8383
ML	0.8360	0.8764	0.8113	0.8602	0.8395	0.8431	0.8470	0.8512
Mejora	-0.0055	0.0313	-0.0122	0.0257	0.0087	0.0132	0.0281	0.0129

Tabla 4.6: Mejora entre escoger el optimizador promedio óptimo y escoger para cada problema el mejor optimizador según el aprendizaje con redes neuronales, para el conjunto de test.

En este caso la mejora es más pequeña en general, incluso empeorando para las muestras 1 y 3. Sin embargo, tomando el promedio de las medidas anteriores vemos que la mejora es positiva,

como se muestra en la Tabla 4.7, en este caso de 0.0128.

Promedio mejor salida	Promedio ML	Mejora
0.8328	0.8456	0.0128

Tabla 4.7: Mejora obtenida al realizar el aprendizaje con regresión lineal

Si calculamos el porcentaje mejorado, respecto al margen de mejora que teníamos anteriormente tenemos

$$\%_{mejorado} = \frac{0.0128}{0.1672} = 7.66\,\%,$$

un porcentaje de mejora menor que con el método de entrenamiento anterior.

Incluimos los gráficos para la muestra 2, que es la que tiene asociada una mayor mejora. Analizando el gráfico de cajas podemos ver de nuevo que la caja asociada al aprendizaje está más cercana a 1 que la del mejor optimizador, Baron (ver Figura 4.3).

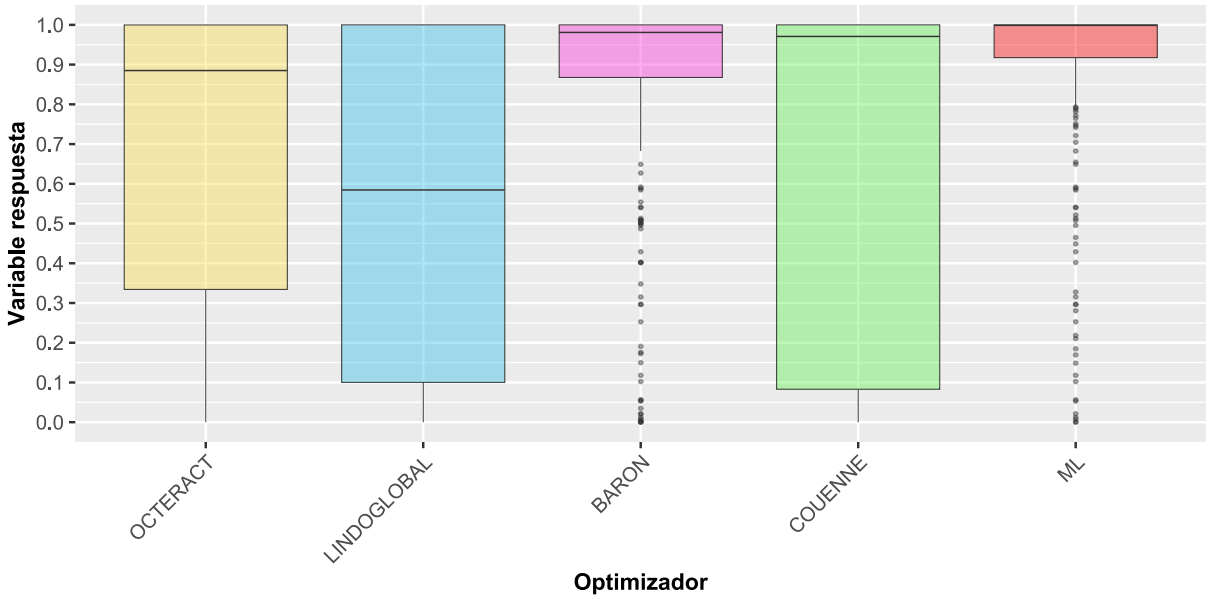
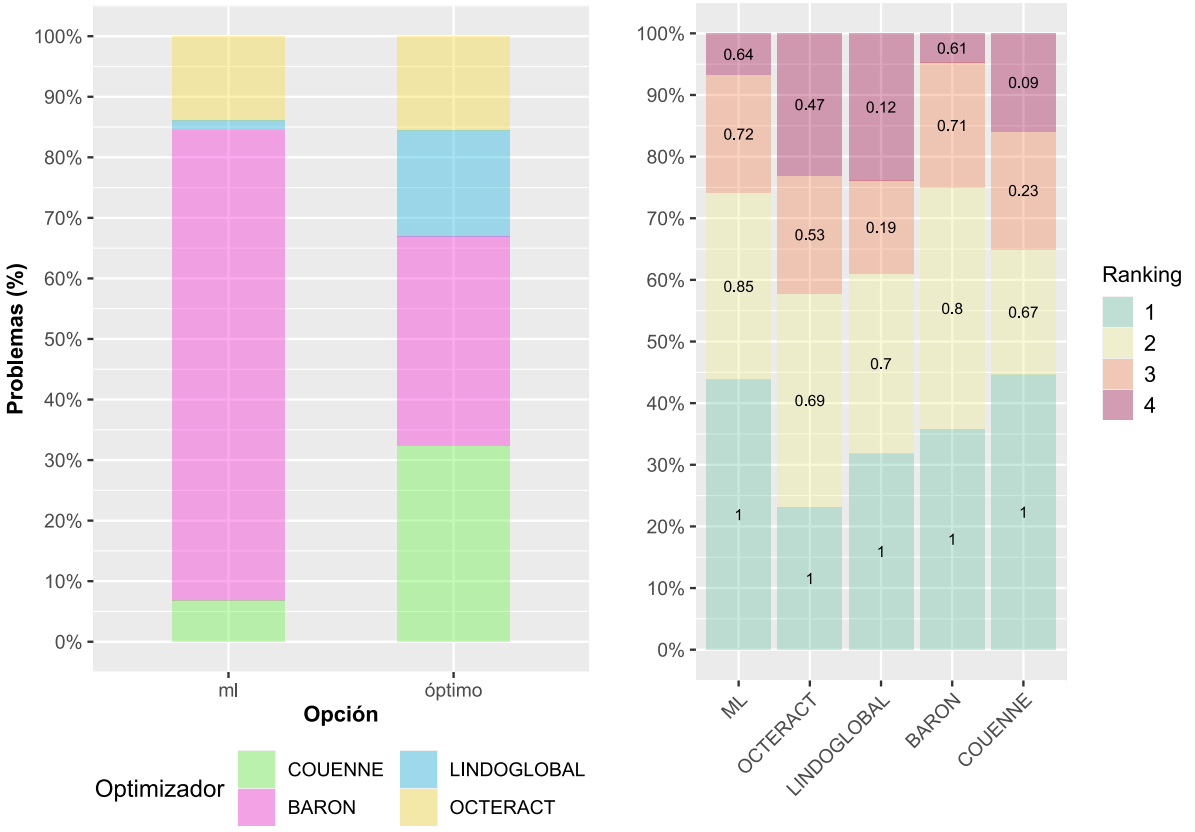


Figura 4.3: Diagramas de cajas para el aprendizaje con regresión lineal.

Si analizamos los gráficos de barras apiladas de la Figura 4.4 podemos ver en este caso que en 4.4a las barras por optimizador son un poco más parecidas que en el caso anterior. De todas formas el aprendizaje sigue tendiendo a tomar Baron como optimizador de referencia en la mayoría de los casos. En 4.4b vemos que para este método de aprendizaje el número de veces que el "ML" acierta es más elevado que en el caso de la regresión lineal. En cambio el número de veces que toma el optimizador peor es algo más elevado. De todas formas, la media de la variable

respuesta para cada uno de los puestos de la columna "ML" sigue siendo la mayor, por lo que el estudio está siendo fructífero y las predicciones son buenas.



(a) Ranking de optimizadores y la variable respuesta asociada.

(b) Comparación entre el aprendizaje y el óptimo real.

Figura 4.4: Gráficos de comparación entre estrategias.

Finalmente haremos una breve comparación entre las dos estrategias de aprendizaje empleadas. Ya vimos que la regresión lineal, un método que destaca por su sencillez, se comportaba bien en general para el estudio que realizamos. Por otra parte, las redes neuronales conseguían unos resultados algo peores, pero afinaban más al elegir la estrategia óptima en algunos casos. En la Tabla 4.8 se puede ver, para cada muestra, la comparativa entre la media de la mejor salida y la salida media con cada estrategia. Solo hay dos muestras (4 y 8) para las que las redes neuronales han proporcionado una mejor predicción, frente al método clásico de regresión lineal. De todas formas, la potencia de las redes neuronales puede ser mucho mayor, y para este trabajo se ha ilustrado una de las formas más sencillas que poseen. Se podrían considerar redes con muchas más capas ocultas y más neuronas por capa, además de emplear funciones de activación no linea-

Muestra	1	2	3	4	5	6	7	8
Mejor salida	0.8415	0.8451	0.8235	0.8345	0.8308	0.8299	0.8189	0.8383
ML con `lm`	**0.8658**	**0.8823**	**0.8416**	0.8543	**0.8535**	**0.8525**	0.8391	**0.8685**
ML con `nnet`	0.8360	0.8764	0.8113	**0.8602**	0.8395	0.8431	**0.8470**	0.8512

Tabla 4.8: Comparación sobre la media de la variable respuesta con las dos estrategias de aprendizaje para cada una de las 8 muestras.

les, que proporcionarían otros resultados. Con todo, para el propósito del trabajo los métodos empleados son suficientes e ilustran la forma de proceder para realizar un estudio de aprendizaje estadístico.

Capítulo 5

Conclusiones y trabajo futuro

A lo largo este trabajo hemos hecho un recorrido por el mundo de la programación no lineal entera mixta y algunos de los métodos de aprendizaje que se pueden emplear para mejorar la resolución de estos problemas con los optimizadores existentes.

Comenzamos haciendo un recordatorio de la programación lineal y entera, vista durante el Grado en Matemáticas en el Capítulo 1. En él introdujimos las bases de la programación matemática general, las buenas condiciones de ciertos problemas, en términos de convexidad del conjunto factible, y explicamos uno de los métodos utilizados para resolver MILPs, el Algoritmo de Ramificación y Acotación. Este método resolvía el problema dividiéndolo en subproblemas más fáciles de resolver, para los que la convexidad del conjunto factible nos asegurada la optimalidad global.

Por su parte, en el Capítulo 2 nos centramos en la resolución de problemas de programación matemática no lineales, pudiendo tener además restricciones enteras. Para ello, sentadas las bases del caso lineal entero, se profundizó en el algoritmo de Ramificación y Acotación Espacial, empleando por muchos optimizadores de carácter global. Centrándonos en el caso de COUENNE, comentamos la metodología seguida, que se basaba en reformular el problema y linealizar las restricciones no lineales, de forma que se conseguía ir aproximando la región factible original por regiones lineales, con un problema asociado más fácil de resolver. En muchos de los casos anteriores era necesario también ramificar, afinando cada vez más la solución del problema. Estos dos primeros capítulos permitieron ver la dificultad de resolución de los problemas no lineales, frente al caso lineal, y el interés por escoger un buen optimizador para resolver cada problema.

Es por ello que el objetivo final de este trabajo era comparar el desempeño de varios optimizadores y crear modelos que fuesen capaces de predecir cuál de ellos sería el óptimo para resolver un problema dado. Para ello, en el Capítulo 3 se introdujo el problema de aprendizaje supervisado y se explicaron dos técnicas sencillas que se emplearían en el capítulo numérico: la

regresión lineal y las redes neuronales de una capa oculta. Aunque hay muchas técnicas que se podrían haber seleccionado se tomaron estas dos por motivos de simplicidad, siendo la primera de carácter más matemático y la segunda estando más relacionada con la inteligencia artificial y el aprendizaje automático.

Seguidamente, en el Capítulo 4 se llevó a cabo la implementación de estas dos técnicas mediante R y la librería *ASML* desarrollada por el CITMAga, en la que pude colaborar durante unas prácticas. Los resultados sobre un conjunto de 836 problemas mostraron que se podía mejorar la resolución, tomando en cada caso el optimizador recomendado por el modelo, frente a tomar siempre el que mejor desempeño global tenía. Aunque hubo mejoras, las técnicas seleccionadas dejaron un margen de mejora, que se podría explotar empleando métodos de aprendizaje más elaborados.

Finalmente, este trabajo constituye un punto de partida sobre un posible estudio futuro en las técnicas estadísticas de selección de algoritmos para optimización. Se podría considerar un número mayor de optimizadores a los ilustrados en este documento, además de emplear otros métodos de aprendizaje más potentes, que seguramente consigan mejores resultados en cuanto a la mejora obtenida.

Anexo I

Resolución de un problema MINLP mediante AMPL

Mostraremos la resolución del problema EJ y su reformulación mediante AMPL y veremos que son problemas equivalentes, al tener el mismo valor de la función objetivo.

Recordemos la formulación de nuestro problema

$$\begin{aligned} \min \quad & -5x_1 - 2x_2 \\ \text{sujeto a} \quad & x_1^3 - x_1^2 + x_2 \leq 4 \\ & x_1 - x_2^2 \leq 0 \\ & -2 \leq x_1 \leq 2 \\ & 0 \leq x_2 \leq 4 \\ & x_1 \in \mathbb{Z}, \end{aligned} \tag{EJ}$$

cuyo código en AMPL se puede ver en la Figura I.1.

```
var x1>=-2, <=2, integer;
var x2>=0, <=4;

minimize z: -5*x1-2*x2;
subject to res1: x1^3-x1^2+x2<=4;
subject to res2: x1-x2^2<=0;
```

Figura I.1: problema.mod.

El archivo de ejecución para resolver el problema con el código anterior se muestra en la

Figura I.2.

```
reset;
model "problema.mod";
option solver couenne;
solve;
display x1, x2, z;
```

Figura I.2: problema.run.

Al ejecutar obtenemos la salida

```
Couenne 0.5.7 -- an Open-Source solver for Mixed Integer Nonlinear Optimization
[...]
couenne: Optimal
x1 = 1
x2 = 4
z = -13
```

donde podemos ver que COUENNE encuentra la solución óptima con $x_1 = 1$, $x_2 = 4$ y un valor de la función objetivo de $z = -13$. En la Figura I.3 está representada la región factible de este problema, junto con la solución óptima encontrada.

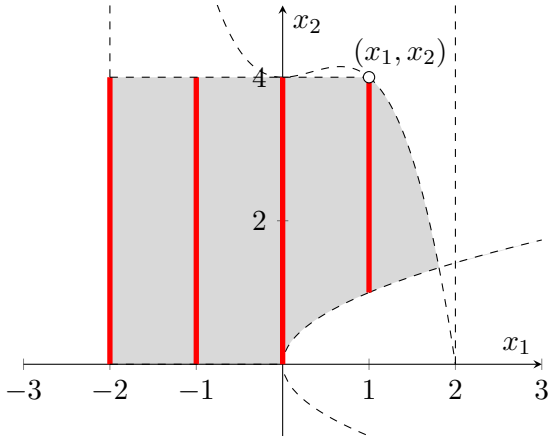


Figura I.3: Solución de EJ.

Procedemos ahora a resolver el problema reformulado, con expresión

$$
\begin{aligned}
\min \quad & x_8 \\
\text{sujeto a} \quad & x_3 = x_1^3 \\
& x_4 = x_1^2 \\
& x_5 = x_2 + x_3 - x_4 \\
& x_6 = x_2^2 \\
& x_7 = x_1 - x_6 \\
& x_8 = -5x_1 - 2x_2 \\
-2 \leq x_1 \leq 2, \quad & x_1 \in \mathbb{Z} \\
0 \leq x_2 \leq 4 & \\
-8 \leq x_3 \leq 8, \quad & x_3 \in \mathbb{Z} \\
0 \leq x_4 \leq 4, \quad & x_4 \in \mathbb{Z} \\
-12 \leq x_5 \leq 4 & \\
0 \leq x_6 \leq 16 & \\
-18 \leq x_7 \leq 0 & \\
-18 \leq x_8 \leq 10, &
\end{aligned}
\tag{EJ'}
$$

para el que tenemos el código del modelo en la Figura I.4

```
var x1>=-2, <=2, integer;
var x2>=0, <=4;
var x3>=-8, <=8, integer;
var x4>=0, <=4, integer;
var x5>=-12, <=4;
var x6>=0, <=16;
var x7>=-18, <=0;
var x8>=-18, <=10;
minimize z: x8;
subject to res1: x3=x1^3;
subject to res2: x4=x1^2;
subject to res3: x5=x2+x3-x4;
subject to res4: x6=x2^2;
subject to res5: x7=x1-x6;
subject to res6: x8=-5*x1-2*x2;
```

Figura I.4: problema_reformulado.mod.

y el archivo de ejecución en la Figura I.5

```
reset;
model "problema_reformulado.mod";
option solver couenne;
solve;
display x1, x2, x3, x4, x5, x6, x7, x8, z;
```

Figura I.5: problema_reformulado.run.

Ejecutando el código anterior tenemos la siguiente salida, donde también se muestran los valores de las nuevas variables definidas

```
Couenne 0.5.7 -- an Open-Source solver for Mixed Integer Nonlinear Optimization
[...]
couenne: Optimal
x1 = 1
x2 = 4
x3 = 1
x4 = 1
x5 = 4
x6 = 16
x7 = -15
x8 = -13
z = -13
```

Vemos efectivamente que las variables x_1 y x_2, que eran las originales de nuestro problema, toman el mismo valor que para el problema sin reformular, obteniendo el mismo valor de la función objetivo.

Bibliografía

[1] Tobias Achterberg, Thorsten Koch, and Alexander Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42–54, 2005.

[2] Sofía Rodríguez Ballesteros. Técnicas de aprendizaje supervisado aplicadas a la resolución de problemas de optimización. Trabajo de Fin de Máster, Máster en Técnicas Estadísticas, 2022.

[3] P. Belotti, J. Lee, L. Liberti, F. Margot, and A. Wächter. Branching and bounds tightening techniques for non-convex MINLP. *Optimization Methods and Software*, 24(4-5):597–634, 2009.

[4] Pietro Belotti, Jon Lee, Leo Liberti, François Margot, and Andreas Wächter. Branching and bounds tighteningtechniques for non-convex minlp. *Optimization Methods & Software*, 24(4-5):597–634, 2009.

[5] Julio González Díaz. Programación lineal y entera. Apuntes del Grado en Matemáticas. USC, 2023.

[6] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. *The elements of statistical learning: data mining, inference, and prediction*, volume 2. Springer, 2009.

[7] A. Khajavirad and N. V. Sahinidis. *A hybrid LP/NLP paradigm for global optimization relaxations*. Mathematical Programming Computation, 2018.

[8] Max Kuhn. The caret package. `https://topepo.github.io/caret/`, 2019. Accedido por última vez 26 de junio de 2024.

[9] Youdong Lin and Linus Schrage. The global solver in the lindo api. *Optimization Methods & Software*, 24(4-5):657–668, 2009.

[10] Mohit Tawarmalani and Nikolaos V Sahinidis. *Convexification and global optimization in continuous and mixed-integer nonlinear programming: theory, algorithms, software, and applications*, volume 65. Springer Science & Business Media, 2013.

[11] Stefan Vigerske. Minlplib. `https://www.minlplib.org/index.html`, 2001. Accedido por última vez 26 de junio de 2024.